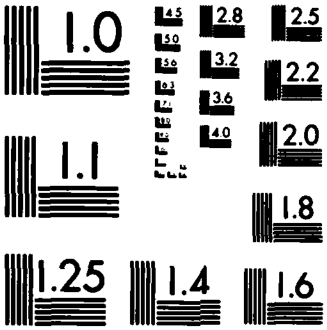
AD-A171 395 MODELLING OF A MULTILEVEL SECURE TACTICAL COMBAT COMPUTER SYSTEM(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA C B CAVALCANTI JUN 86 1/2

UNCLASSIFIED F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

2

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

MODELLING OF A MULTILEVEL SECURE
TACTICAL COMBAT COMPUTER SYSTEM

by

Claudio Augusto Bailly Andersen Cavalcanti

June 1986

Thesis Advisor: Uno R. Kodres

Approved for public release; distribution is unlimited

AD-A171 395

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY 2b DECLASSIFICATION / DOWNGRADING SCHEDULE	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL (If applicable) 52	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5100	7b. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5100
8a. NAME OF FUNDING / SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)

MODELLING OF A MULTILEVEL SECURE TACTICAL COMBAT COMPUTER SYSTEM

12 PERSONAL AUTHOR(S)

Cavalcanti, Claudio B.

13a TYPE OF REPORT Master's Thesis	13b TIME COVERED FROM ________ TO ______	14 DATE OF REPORT (Year, Month, Day) 1986 June	15 PAGE COUNT 121

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Secure System, Multilevel System, Tactical System

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This work is an analysis of the use of a multilevel secure computer system to execute tactical combat applications programs. Using the Gemini Trusted Multiple Microcomputer Base, currently under evaluation by the Department of Defense Computer Security Center, applications and test programs were written and implemented in order to expose some characteristics of the system. Using a Janus/Ada compiler with the necessary library alterations for the Gemini machine, a simple weapons application program was implemented in a system designed to simulate a tactical environment where classified material can be handled in spite of the different levels of security held by the operators that can access the system. The loss in performance due to the secure operating system's overhead is estimated in order to establish the tradeoffs in performance gains due to parallel processing capability of the multiprocessor system.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a NAME OF RESPONSIBLE INDIVIDUAL Uno R. Kodres	22b TELEPHONE (Include Area Code) (408) 646-2197	22c OFFICE SYMBOL 52Kr

83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

Modelling of a Multilevel Secure Tactical Combat Computer System

by

Claudio Augusto Bailly Andersen Cavalcanti
Lieutenant Commander, Brazilian Navy
B.S., Escola Naval, 1970

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ENGINEERING SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June, 1986

Author: ________________________
Claudio Augusto Bailly Andersen Cavalcanti

Approved by: Uno R. Kodres
Uno R. Kodres, Thesis Advisor

Gary S. Baker
Commander Gary S. Baker, Second Reader

____
Vincent Y. Lum, Chairman, Department of
Computer Science

____
John N. Dyer, Dean of Science
and Engineering

ABSTRACT

This work is an analysis of the use of a multilevel secure computer system to execute tactical combat applications programs. Using the Gemini Trusted Multiple Microcomputer Base, currently under evaluation by the Department of Defense Computer Security Center, applications and test programs were written and implemented in order to expose some characteristics of the system.

Using a Janus/Ada compiler with the necessary library alterations for the Gemini machine, a simple weapons application program was implemented in a system designed to simulate a tactical environment where classified material can be handled in spite of the different levels of security held by the operators that can access the system.

The loss in performance due to the secure operating system's overhead is estimated in order to establish the tradeoffs in performance gains due to parallel processing capability of the multiprocessor system.

THESIS DISCLAIMER

The reader is cautioned that computer programs developed during this research are not completely validated. Although every effort has been made in order to make the programs free of computational and logical errors, the time available was not sufficient to perform a fully reliable job. Documentation and software used in this work were supplied by the manufacturers of the microcomputer used, in a preliminary format with updates being received throughout the research period.

Some terms used in this thesis are registered trademarks of commercial products. All trademarks appearing in this thesis will be listed below following the firm holding the trademark:

1. Gemini Computers Inc. Monterey, California.
   Gemini Trusted Multiple Microcomputer Base.
   GEMSOS
2. RR Software Inc. Madison, Wisconsin.
   Janus/Ada development package.
3. INTEL Corporation, Santa Clara, California.
   INTEL
   Multibus
   APX-286

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. PROBLEM STATEMENT

This thesis investigates the use of a multilevel secure computer system in a tactical combat environment. The specific application of the system proposed is to perform the duties of a real-time system with the extra ability to handle sensitive information in a trusted manner.

A real-time system is defined as:

> A system that reacts as to affect the environment in which it is operating. It is a collection of devices, controlled by a stored program of instructions. This program acts as the regulating element in a feedback loop, which then forms part of a system. [Ref. 1: p.1]

Sensitive information is defined as a collection of data that cannot be accessed but by those who have specific authorization.

The type of environment where a secure tactical system can be implemented is shown in Figure 1.1 , which depicts a hypothetical section of the operation's room of a combat ship.

The tactical program executed by this specific system requires some secret data, in order to produce the desired results. The system should allow the tactical program to access the secret data, make the necessary computations and transfer the result to the desired peripheral. The operator who "drives" the tactical program should not have direct

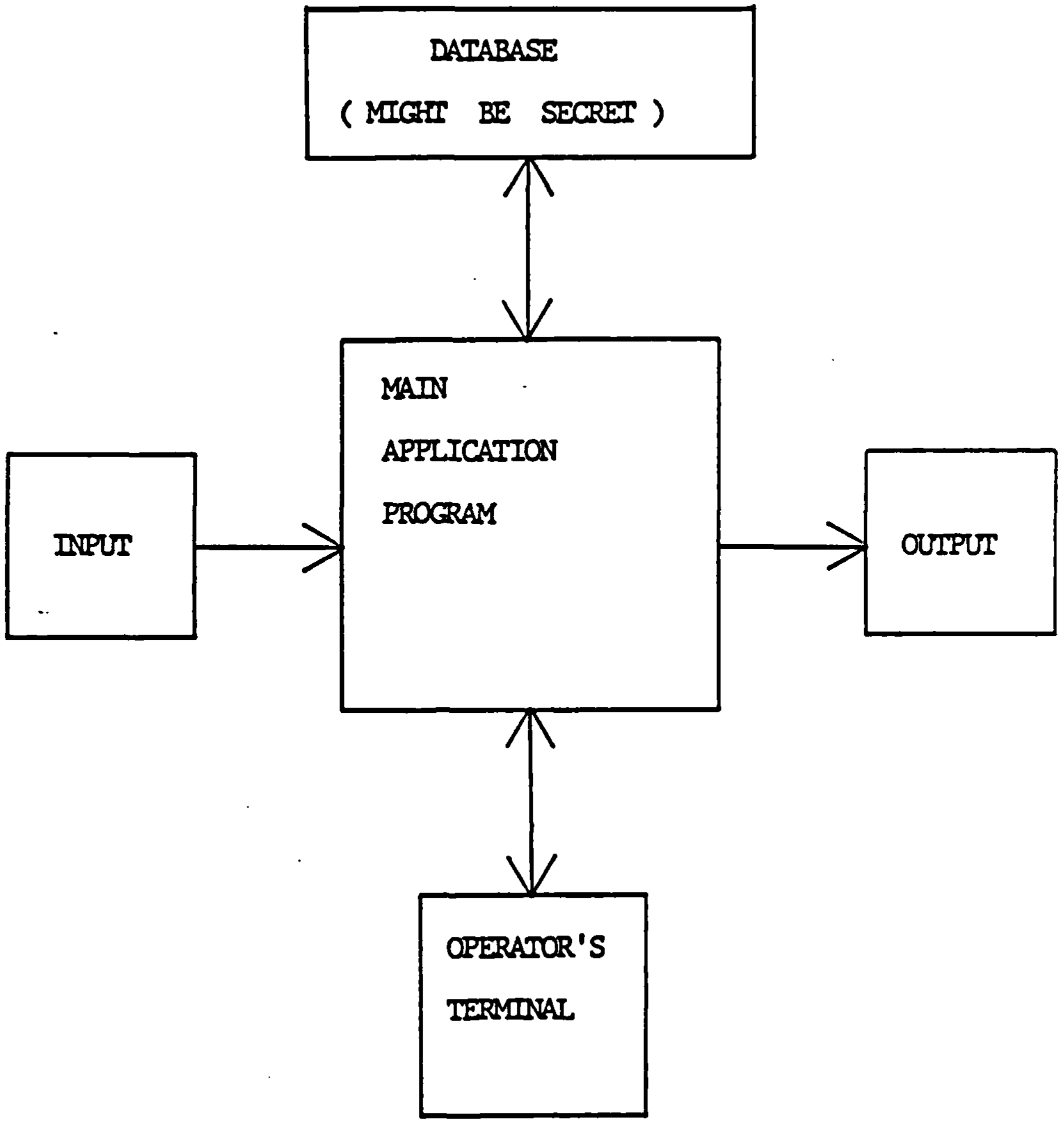


Figure 1.1 A Tactical Secure Environment

access to the secret data. The eventual access to the secret data, i.e. to update some parameters, should be allowed only to operators with secret clearance or above.

Another important aspect of a tactical secure combat system is that the tactical program needs to be maintained by on-board technicians, who might not have secret clearance. If a "free" access to the secret data is allowed, a skillfull maintainer with some corrupt intentions, can easily produce a "patch" which will extract the sensitive information and transfer to a printer or a display. This dangerous picture is very much likely to exist and in fact, such a problem ocurred two years ago on board of an aircraft carrier, involving the geographic positions of U.S. Navy's nuclear submarines.

In order to avoid the necessity to clear all the maintainers on-board to secret level, some sort of independence should exist between the modules of a tactical program. The modules should be able to receive different security labels that cannot be changed by on-board maintenance. The modules authorized to be maintained on board should be re-integrated into the system with no changes to the security parameters.

Since tactical environment automatically calls for speed, all the necessary security techniques must not create too much overhead to the overall performance.

This research was performed in conjunction with the Naval Postgraduate School's AEGIS Modelling Group. This group is sponsored by the AEGIS Combat System Project Office to conduct research in the area of combat system development.

To summarize the problems discussed in this section we can state the following requirements for a tactical secure combat system:

1) The system will execute tactical programs that uses sensitive information;

2) The access to the sensitive information should be controlled;

3) The tactical programs need to have an on-board maintenance by technical personnel with no clearance to the sensitive information;

4) Changes to the security parameters should not be possible unless by authorized personnel;

5) There should be no large overhead due to the security aspects of the system.

B. PROPOSED SOLUTION

This thesis proposes the use of a multilevel secure computer system to execute the tactical combat program. The secure computer system would be the "heart" of the proposed system, executing the application program specifically designed for each different situation. The multilevel secure computer system, based upon the security level of the operator currently logged on, would perform different kinds of functions. The capacity of labeling the modules of execution in a multilevel secure computer system would allow the isolation of sensitive modules, thus protecting them against unauthorized users.

An schematic view of the proposed solution is shown in Figure 1.2.

Here, an application program would be delivered to the ship's system containing five independent modules. Module 1 is the "master" module, which controls the execution of the whole system and performs the synchronization between modules. It would not be permitted to maintain this module on-board. Modules 2 and 3 contain algorithms to process inputs and outputs and some intermediate calculations. Those modules can be maintained on-board. Module 4 accesses the secret data, and cannot be maintained on-board. Module 5 contains the secret data, and cannot be maintained on-board although alteration of some specific fields can be done by an authorized operator.

When an unclassified operator logs on, the master module will activate modules 2 and 3, which will execute and call module 4 to access the secret data. The secret data is then, handled only by module 4 which provides the result of the operation requested by modules 2 and 3, but will never transmit the information read from module 5.

When a "secret" operator logs on, the master module will activate directly module 4, which accesses module 5, but this time the information read is transferred to the

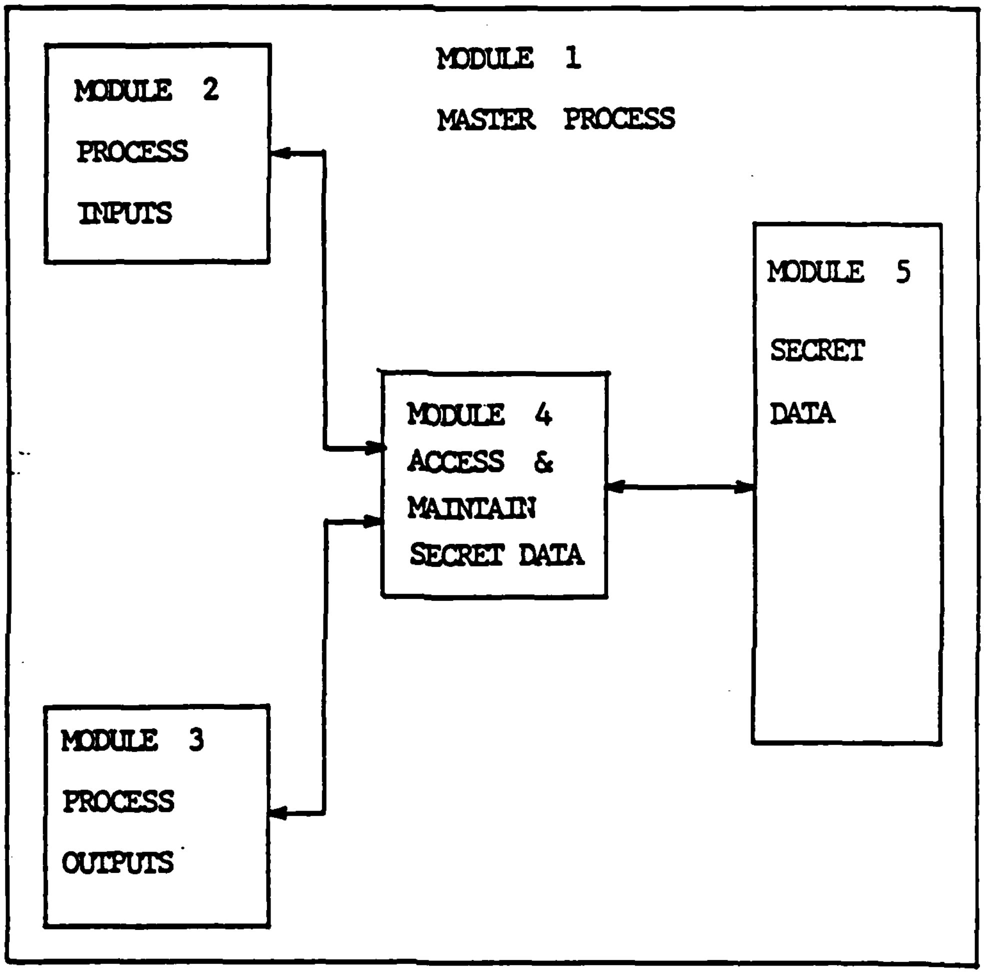


Figure 1.2 A Secure Processes Interaction

operator whose "secret" classification level authorizes access to the information.

There are several systems being currently evaluated by the Department of Defense to operate as a multilevel secure system. The Gemini Trusted Multiple Computer Base is the one used in this research. This system is still undergoing development and so, some restrictions were imposed.

C. THESIS FORMAT

This thesis is composed of five chapters ordered in a sequence to provide the reader with a presentation of the problem, some background information in secure systems and then introduce the design and the implementation of a system to execute tactical secure combat programs.

Chapter I presents the problem and the solution in a generic format.

Chapter II describes the concepts of multilevel security and provides some detailed information about the microcomputer used in this research, the Gemini Trusted Multiple Computer Base.

Chapter III discusses the actual design of the application program for the proposed system.

Chapter IV covers the implementation and testing of the application program using the Gemini system. Some information about loss in performance due to overhead caused by security checks is included in this chapter.

Chapter V analyzes the results, proposes some techniques for the development of applications programs and suggests some follow-up research.

II. SECURE SYSTEMS

A. TRUSTED COMPUTER

1. Background

There is not to date an unique and generally accepted definition for a trusted computer system. Depending on the origin (i.e., business or government), the requirements can be quite different and sometimes they even conflict between each other. The Department of Defense, with the purpose to define parameters for any work in the area of secure systems, has elaborated a document which is entitled DOD Trusted Computer System Evaluation Criteria and was published in 1983 [Ref. 2]. This document established guidelines for the test and evaluation of any new system involving security aspects. It contains all the information necessary to anyone involved in research with trusted computer systems.

One of the important concepts described in this publication, which has direct effect in the analysis executed in this thesis, is the establishment of two basic types of security policy: Mandatory (sometimes called Non-discretionary) and Discretionary Security.

Mandatory Security is defined by the following direct quote.

> Security policies defined for systems that are used to process classified or other specifically categorized

> sensitive information include provisions for the enforcement of mandatory access control rules. That is, must include a set of rules for controlling access based directly on a comparison of the individual's clearance or authorization for the information and the classification or sensitivity designation of the information being sought, and indirectly on considerations of physical and other environmental factors of control. The mandatory access control rules must accurately reflect the laws, regulations, and general policies from which they are derived. [Ref. 2: p.72]

As it can be understood from the definition above, the mandatory policy is expressed by a lattice of access classes. The mandatory policy establishes that the control of the accesses is based on an access level determined by the user's security clearance and this policy cannot be modified or bypassed within the system. The mandatory policy, furthermore, establishes the limits for the discretionary security, which is an additional set of constraints on access to information, based on some particular constraint, like the military "need-to-know" policy.

The Discretionary Security is the second type of security policy established by the DOD document and it is expressed by direct quote which follows.

> Security policies that are defined for systems that are used to process classified or other sensitive information must include provisions for the enforcement of discretionary access control rules. That is, they must include a consistent set of rules for controlling and limiting access based on identified individuals who have been determined to have a need-to-know for the information. [Ref. 2: p. 73]

There will be certain situations where although the elements of a group, involved in an analysis or a research project may have the same clearance, the manager wants to limit the type of information which each one should access. This can be done in order to extract different and unbiased opinions and observations. The discretionary policy is the tool to provide this access granularity without affecting the mandatory rules.

2. The Threats

The attractive field of computer technology has had in the past few years one of the fast and impressive develtopments ever observed in a new science. This has lead to a proliferation of computers and networks so that it is improbable to find today any company or organization that does not use some kind of a computer system.

Among all the information stored, some is classified, and so, need special care. The few and basic security controls first used were considered sufficient to limit the access to classified information. The machines were physically isolated and locked. Some more sophisticated systems had a software coded control. As it has always happened in human history, there is always a conflict of interests and there is almost no limit to the human desire and dedication. So, the computer "hackers" entered the scene and there has been a lot of break-ins widely reported by the press, in many types of computer systems. The control processes had to be improved and the break-in techniques improved concurrently!

The only very low break-in probability technique ended up to be the enclosure of the peripheral, which permits access to the classified information, in a tightly secured vault. Evidently this is not a satisfactory solution. In some tactical applications, for instance, the operators might have no secret clearance, but the data the tactical program uses, is secret. A bright and "interested" operator can use this situation to create a software patch, for example, which although transparent to the normal operation of :the system, will extract some of the classified information that is stored somewhere.

3. Proposed Technology

Research centers and universities have conducted a great amount of work and fortunately, some very good results are now available to be implemented. The security kernel technology [Ref. 3] has been considered the driving force for the building of trusted computer systems and several products have implemented and improved this technique in an effort to turn the products into practical, simple to use, and most of all, secure systems.

To determine if a system is secure or not, is a very difficult task, starting with the problem to establish the criteria to evaluate the performance. The Departament of Defense is preparing a document which will contain the details of such an evaluation criteria and this analysis is not considered in this work.

The employment of secure products by potential customers is usually not considered until some harmful break-in happens, mainly because the practicality of its use has not been demonstrated yet. In the tactical environment there are extra concerns like timing, adaptability and real-time applicability that have to be demonstrated together with the secure capabilities, in order to integrate these products into a combat system.

The Gemini Trusted Multiple Microcomputer Base using the Gemini Multiprocessing Secure Operating System (GEMSOS) is claimed by its manufacturers to fulfill the tactical requirements with secure aspects, and this was the machine used in this research.

B. THE GEMINI SYSTEM

1. General

The Gemini Multiprocessing Secure Operating System (GEMSOS) was designed for the Gemini Trusted Multiple Microcomputer Base, in order to have the system to operate at the B3 [Ref. 2] level of classification, although the ultimate goal is to meet the class A1, the highest level defined. The system is currently under evaluation by the Department of Defense Computer Security Center for certification to the B3 class. The system was developed based on the security kernel technology [Ref. 3] like all trusted systems are, and the main idea was to provide an off-the-shelf product, using state of the art, hardware components and software engineering techniques. The main characteristics of the system are [Ref. 4]:

1) Can operate with up to eight Intel APX-286 based microcomputers in parallel, giving a great processing power. The microcomputers communicate through shared memory segments providing high throughput, and the GEMSOS minimizes bus contention by locating data and code in the local memory of each processor, whenever it is possible.

2) The multiple microcomputers are capable of multiprocessing and multiprogramming. The GEMSOS can multiplex processes to a single processor or distribute processes to several processors, so that both parallel and pipeline processing is possible.

3) Concurrent computing is independent of the programming language used, since the GEMSOS provides its own primitives to manipulate the abstracts <u>eventcounts</u> and <u>sequencers</u>, in order to support communication and synchronization among processes.

4) A variety of I/O devices and storage, which include fixed disks, high density floppy diskette drives and non-volatile memory, can be directly connected to the Multibus. Each RS-232 interface board can handle up to eight devices.

5) The system includes some other features like a real time clock, data encryption device (NBS-DES algorithm), a system unique identifier to prevent covert channels, and a non-volatile memory to store passwords and encryption keys.

The Gemini system allows the development of applications programs using theoretically any language supported by the CPM-86 operating system. Some additional files, which will change the utility library associated with the programming language, has to be provided with the Gemini system. As the decision was to generate application software using the Janus/Ada computer language, the actual coding of this work had to wait the delivery of the Janus/Ada environment software and documentation, which happened in late March. There are special features for the Gemini Janus/Ada environment that cannot be easily adapted from a normal Janus/Ada code and these will be pointed out later in this work. The current implementation of Janus/Ada on GEMSOS still does not include the ability to use a Janus/Ada process as the initial Ring 1 process and some limitations result from that.

The claimed ability to handle different hardware configurations is an important characteristic of the Gemini system. If the system is going to be used in real tactical applications, this certainly is an important aspect.

2. Resource Management Concepts

A set of resource management services that can be invoked by an application program is provided by the GEMSOS, in order to provide the customer with tools to control the performance of the particular implementation under development. The application program uses what is called a service call, which can be treated as a subroutine call, with arguments being passed and returned. These service calls are called gate calls, since they make certain security checks to allow the flux of data. The actual details of each call is specific to the language being used and is supplied in the GEMSOS interface library provided with each compiler (Janus/Ada in this study).

The GEMSOS kernel is divided into three basic management areas, which are: segment management, process management and device management.

a. Segment

Segments are discreet and logical objects (entities) that contain all the information to be manipulated in a Gemini system. The segments of concern to the applications programmer are the code, data and stack segments.

The GEMSOS kernel allows some application program to move data within the system in such a way as to be immediate available to a particular process or not. There are eight different calls provided for this management. These calls will handle the movement, creation and termination of data as well as the transfer of the necessary information to the Kernel's mandatory security model, which will deny or accept the request for service. The segment manager controls a "Known Segment Table" (KST), where the segment numbers are related to the system-unique identifier of the segment usable by the memory. The segment when created, will receive a tag associating it to a particular collection of segments, called a volume, which is the unit of secondary storage. A volume can be treated as separate entity and so be called by a process. A detailed description about each of the segment management calls is contained in [Ref. 5].

b. Process

The management of a process includes the actual management and the synchronization between processes.

(1) Management. When created, each process is uniquely identified by code, stack and data segments and at the same time, a fixed amount of resources is assigned to it. There are four primitives to manage a process.

(2) Synchronization. Once a segment is created in an application program, an eventcount and a sequencer are automatically associated with it. These two abstract objects have the same name as their owning segment. The process can then be synchronized with other processes by means of four primitives supported by the kernel, which are: advance, await, read and ticket.

c. Device

The Gemini system treats the management of devices in a very peculiar way, which is, to reside most of the functions dealing with I/O management in the code at the application level. This design is two-fold. It reduces the size of the kernel making verifications easier, but it also makes the I/O applications software more difficult to be coded. There are six calls to handle a device. The I/O device controller is treated as a process, which is then synchronized with the segments eventcounts and sequencers to perform the desired functions. More information about device management, process management and synchronization, can be found in [Ref. 5].

3. The Operating System

The Intel APX-286 supports four protection levels and GEMSOS uses them as four hierarchical rings to enforce the security layering. They are numbered from 0 to 3, 0 being the most privileged one. The mandatory and discretionary policy are supported in rings 0 and 1 respectively. The mandatory policy, as already mentioned, cannot be modified and is represented as a lattice of access classes in the distributed kernel contained in ring 0. This distributed kernel in ring 0 will virtualize processors, storage, I/O and objects (processes, segments and devices). Ring 1 supports the discretionary policy and any other security requirements. The supervisor, which is built on the kernel, uses the virtualized objects to perform the normal functions of an operating system. The other two rings, 2 and 3, are used by the programmers for the development of applications.

The implementation of a reference monitor [Ref. 3] is the base of the GEMSOS. All the access by the active entities, subjects, to passive entities, objects, has to be mediated by the reference monitor as shown in Figure 2.1.

All these checks are performed by the security kernel located in ring 0. The subjects are processes allowed to perform in a specific domain, and objects are pieces of information that are observed or modified. Both have security labels assigned to them. The result of the comparison between the security labels of the subjects versus the objects, is what decides if the transaction is approved.

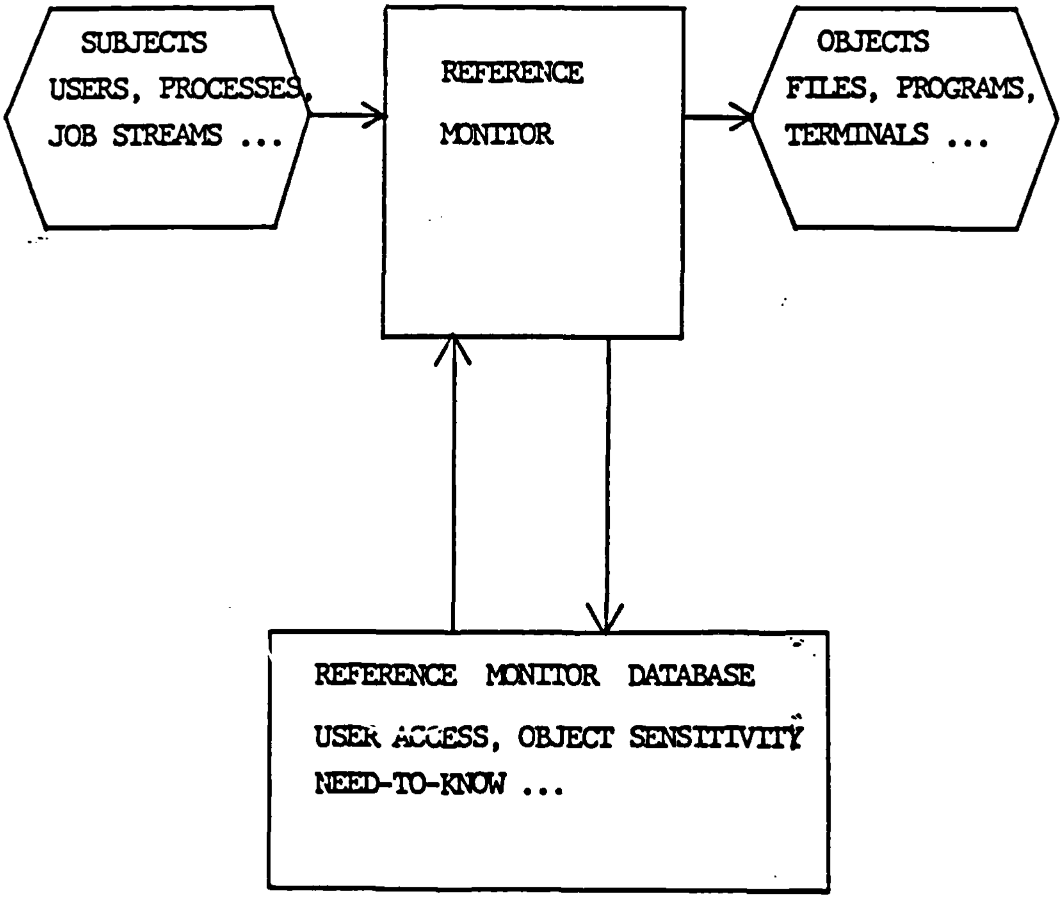


Figure 2.1 The Functions of a Reference Monitor

Security label is a tag that represents the access class of an entity. This access class is defined as having two components: a compromise level and an integrity level. There are properties that establish the criteria for an access to be granted based on the compromise and integrity protection enforcement rules. These properties are listed in [Ref. 4: pp. 16,17] and are summarized here as follows:

Compromise Properties.

1) If a subject has "observe" access to an object, the compromise access component of the subject must dominate the compromise access component of the object.

2) If a subject has "modify" access to an object, the compromise access component of the object must dominate the compromise access component of the subject.

Integrity Properties.

1) If a subject has "modify" access to an object, then the integrity access component of the subject dominates the integrity access component of the object.

2) If a subject has "observe" access to an object, then the integrity access component of the object dominates the integrity access component of the subject.

Compromise can be related to the secure distribution of information, and integrity to the secure modification of information. "Dominates" in the above properties, means access level greater than or equal to the referred entity.

The number 1 property of both compromise and integrity protections are the traditional security policies which are called simple security properties. They state that in order to observe/modify some information one has to have a clearance at least equal to or greater than the information referenced.

The number 2 property, on the other hand, is not usual and it is called the *-property (star property). The purpose of this protection is to avoid an indirect observation or modification of an entity by an "inferior" one. In the compromise situation, for example, a secret process could modify an unclassified file if this protection did not exist. This "modification" could easily be the transmission of secret data that the secret process has access, to the unclassified file. In the integrity situation, it prevents a secret process of observing an unclassified file, and this observation could be "read some data and include it in your computation", which will allow the secret process to be influenced by an unclassified user. In [Ref. 6] there are some more comments about the types of attacks (Trojan Horse) that can result if these properties are not enforced.

Ring integrity is enforced, in addition to all those properties already mentioned, in the Gemini system. It means that, subjects can only access objects with equal or greater ring number, which enforces the hierarchical structure of the rings.

The rigid observance of the properties mentioned above, would transform the simple task of distributing messages (when they have different access classes), into a very complicated and resource consuming procedure. As the Gemini system is a multilevel system, this would be the case. In order to avoid this problem, the *-property for compromise and integrity are relaxed within a certain range of security levels. The process, which has certain flexibility in order to execute some trusted activities, is called "trusted" subject, and it is up to the application programmer that his "trusted" process does not violate the security policies. In GEMSOS, the implementation of "trusted subjects" are in the form of multilevel subjects and they are trusted within a range, demarcated by their maximum and minimum access classes. As mentioned already, only subjects guaranteed not to improperly observe or modify information, should be created as multilevel subjects. Extreme caution should be emphazised when interfacing with devices.

The range of access classes for devices, should be chosen depending on the physical location in which they operate. Devices can be single level or multilevel, and the classification is based on the data they manipulate,whether they have a security label attached to it or not.

The security properties of single and multilevel devices are the following [Ref. 4: pp. 21,22]:

Single-level Devices.

1) To receive ("read") information:
Process maximum compromise }=Device minimum compromise
Device maximum integrity }=Process minimum integrity

2) To send ("write") information:
Device maximum compromise }=Process minimum compromise
Process maximum integrity }=Device minimum integrity

Multi-level Device.

1) To receive ("read") information:
Process maximum compromise }=Device maximum compromise
Device minimum integrity }=Process minimum integrity

2) To send ("write") information:
Device minimum compromise }=Process minimum compromise
Process maximum integrity }=Device maximum integrity

4. The NPS configuration

The Gemini system used during this research has the following configuration:

1) one Intel APX-286 microcomputer

2) two 1.2 Mbyte floppy disk drives

3) one RS-232 interface board with a maximum of eight ports

This system proved to be sufficient for the execution of some preliminary processes like the ones presented in this thesis. However, the amount of time expended during compilation, linking and sysgening and the constant swapping of floppy disks due to the floppy disk drive environment was a big constraint.

III. TACTICAL SYSTEM DESIGN

A. DESIGN ISSUES

1. Objectives

The primary objective of this design was to develop a model which would demonstrate the use of the Gemini Trusted Multiple Computer Base in a tactical combat environment. Based on the requirements for a tactical secure combat environment listed in the introductory chapter, the model shown in Figure 3.1 is presented.

In this model, the Gemini computer would be used to receive the encoded data from a tracker radar, and transmit some positioning information to a weapons device. The actual devices being controlled in this model, are irrelevant at this point. The application program executed by the Gemini computer would make use, for the computation of the results, of some stored information classified as secret. This can be better understood, if we suppose that the tracker radar is tracking an incoming missile, and the desired response, is the firing of a chaff burst as a defensive procedure. During the computation phase, the program has to access data about our own ship which might be classified. The operator controlling the tactical picture cannot have direct access to his data. However this data has to be updated eventually by some authorized operator.

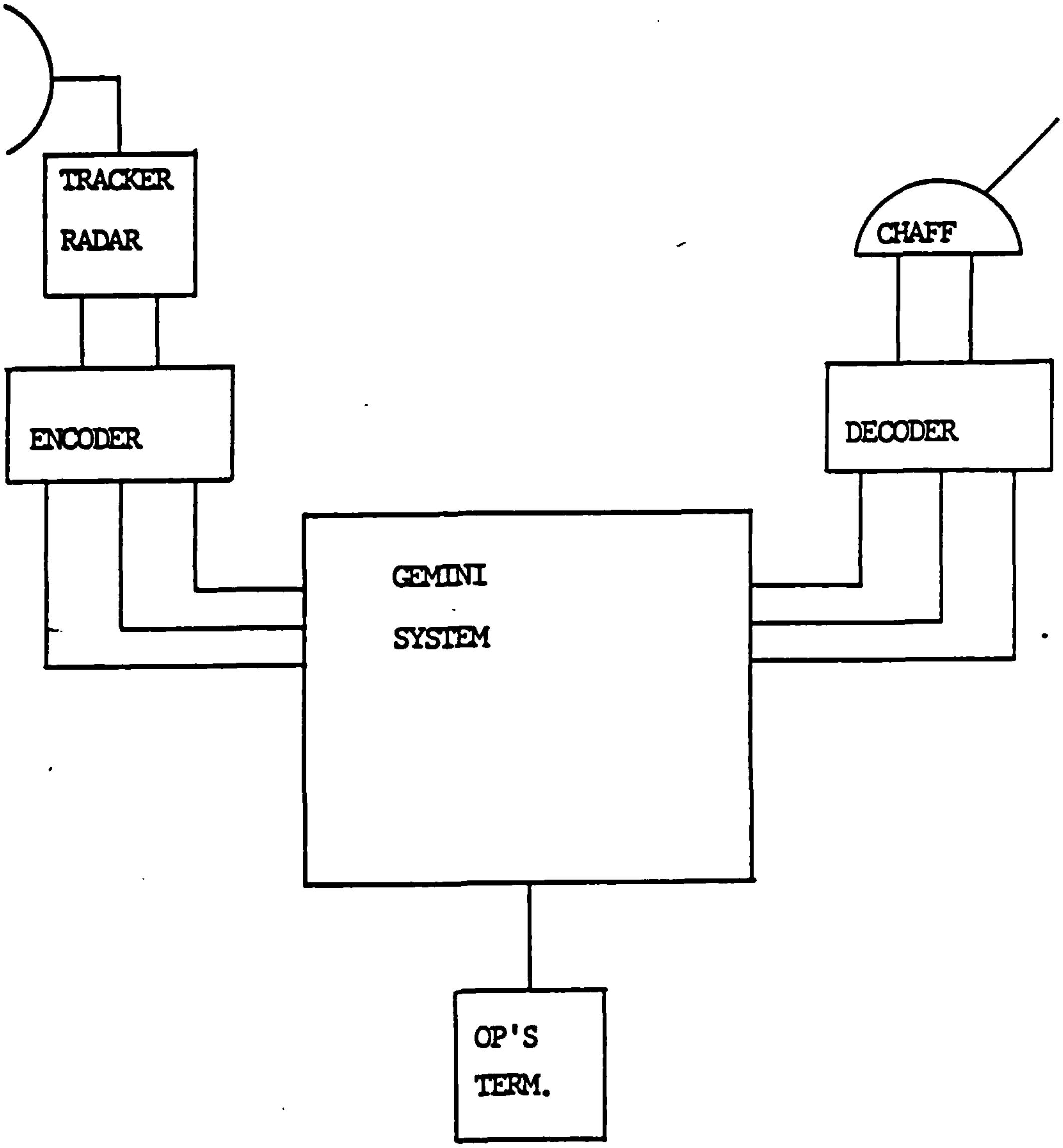


Figure 3.1 Tactical Combat Model

In addition to all that, the system has to have a very fast response, and the steps of execution (reception of radar data, computation of results, transmission of position) have to be precisely synchronized.

2. Hardware Simulation

The construction of the model described in the previous section, would be ideal, since the attachment of different devices to the Gemini computer would be tested. Response time, encoding input techniques and many other aspects would be revealed. This should be part of a follow-up research.

As a preliminary research, the development of the application program was considered the main task.

The complete model, will then be simulated as follows: One terminal attached to a serial port is going to simulate the radar input. The values sent to the "main" process will be generated by software. Another terminal will perform the same simulation of the weapon to be driven, showing on the screen the transmitted values. The "main" process will make use of the secret data stored in another segment, simulating a secondary storage.

These simulations will not disturb the development of the application program. The processes to be created in order to perform the simulations described above, would be necessary in the complete model as well. The main difference would be in the code itself, since the processes would be executing controlling functions.

B. SOFTWARE DESIGN

The application software for this system was designed using the modular programming construction technique. In the particular case of the system used, which had a floppy disk environment, this technique was very useful, because the modules could be compiled separately. Unfortunately, the testing of the modules cannot be done separately, when the modules execute calls to the GEMSOS. To prepare a program to be executed in the Gemini computer, which is going to be explained in a further section, takes about 15-17 minutes, and the main process (the parent process) has to be included always, since the creation of processes and synchronization are coded in the main process.

The application software was then divided in four application programs:

1) "THEMAIN", the parent process, containing the initialization, creation of processes, synchronization and deletion of processes(log off).

2) "RADAR", a child process, performing the simulation of the radar inputs, and the transmission of data to the parent process.

3) "COMPUTE", a child process, which receives data from the "RADAR" process, execute some computations, and tramsmit the results to the "CHAFF" process.

4) "CHAFF", a child process, which will receive data from the "COMPUTE" process, and will simulate the positioning of a weapon.

1. The Parent Process

This is the controlling process for the whole system. The creation of the child processes is established in this module, together with the security parameters and the synchronization scheme. This module has to be designed and coded by a programmer cleared to the maximum level of security to be used, since he will decide the levels for each of the child processes to be created. The coding of the child modules, will be given to different programmers, depending upon the security level of the module.

The general algorithm for a parent process is shown in Figure 3.2.

```
Package body THEMAIN is

begin
    perform initialization;
    create segment to be parent;
    create segment to perform synchronization;
    create processes;
    loop
       call child 1;
       call child 3;
       call child 2;
       exit when some condition;
    end loop;
    delete processes;
end THEMAIN;
```

Figure 3.2 Package THEMAIN

There are some other procedures, to transfer data to and from the child processes, not shown here. They can be found in the program listing in Appendix A.

2. The Child Processes

These processes will perform some defined function which has been determined by the software manager. The actual details of implementing the code are left to the programmer in charge.

In our application program, the child modules execute the general algorithm described in Figure 3.3.

This is just a general algorithm, and the full listing of each module used in the application program developed in this research, is shown in Appendix A.

```
Package CHILD body is

begin
    receive data from parent;
    perform calculation;
    execute simulation;
    pass data to parent;
end CHILD;
```

Figure 3.3 Package CHILD

As it was mentioned before, the child processes can be maintained separately, as long as the synchronization part of each module is not changed. If a module is to be labelled as secret, the maintenance can be restricted to Authorized personnel. The preparation of the complete program to be executed in the system, will be done by a user with the necessary level of security.

C. SOFTWARE DESIGN GOALS

The configuration used for this research had some restrictions, as already mentioned, and among them, the amount of time necessary for each development step represented considerable difficulty. The GEMSOS calls using the Janus/Ada language are yet under development. A preliminary version of the Janus/Ada software library and manuals were received in March 86. Due to these reasons, the following sequence of steps has been established for the development of the application program:

1) Demonstrate the attachment and detachment of a terminal.

2) Demonstrate an application program which samples an input device, performs calculations, and presents the result, all synchronized sequentially.

3) Extract some information about overhead caused by some GEMSOS calls.

4) Synchronize the application program via a real-time clock.

5) Label one of the child processes as secret, and test the access for different operators.

This research was performed in cooperation with another student, Major Miguel Reyes, Peruvian Air Force, who has one more quarter to work on this system. Hopefully, the steps not accomplished by this thesis would be demonstrated in his work.

IV. IMPLEMENTATION ON GEMINI SYSTEM

A. GEMSOS LIBRARY

When developing Janus/Ada application programs to run on GEMSOS, the standard I/O and file type utilities provided with the normal Janus/Ada compiler, cannot be used. Instead, the GEMSOS gate calls provided by the manufacturers, have to be used. As the Janus/Ada environment provided for use with the GEMSOS, is not complete yet, some of the utilities necessary for the development of the application program had to be constructed. Four packages were built to modularize the procedures, functions and declarations necessary for the present application: MANAG, GEMIO, CRPROCE, TABLES.

1. Package MANAG

This package includes the procedures necessary for the management of segments and terminals. To create a segment, a number of parameters should be passed to the GEMSOS call CREATE_SEGMENT. These parameters are then explicitly passed in this procedure.

Any device to be used by the Gemini system needs to be attached. This applies to the screen terminal as well. The GEMSOS call ATTACH_DEVICE, has a specific configuration parameters which are used with terminals. The same applies for the GEMSOS call DETACH_DEVICE. Two procedures were then built, in such a way that some parameters which are constants for terminals do not have to be passed.

2. Package GEMIO

This package is designed to be the I/O package for the Gemini system. The procedures included here, are the ones found to be necessary up to this point of the research. Evidently, many more have yet to be developed, in order to have a comprehensive I/O package. Some of the procedures included in this package were taken from the demonstration program supplied by the manufacturers of the Gemini computer.

3. Package CRPROCE

In order to create a process, the general algorithm presented in Figure 4.1 has to be applied.

```
Package body CREATE A PROCESS is

begin
    makeknown the mentor segment
    specify the address for the process stack
    specify the address for the process code
    specify the address for the process mentor
    specify the address for the trap segment
    calculate the stack size
    create the segment for the stack
    makeknown the segment for the stack
    swap in this segment
    create the segment for data
    makeknown the segment for data
    swap in this segment
    complete the record for the CREATE_PROCESS call
    call the GEMSOS CREATE_PROCESS
end CREATE A PROCESS
```

Figure 4.1 Package CRPROCE

As it can be seen from Figure 4.1, the creation of a process involves a large number of steps. The size of this procedure alone, justifies the construction of a package containing just this procedure.

4. Package TABLES

The purpose of this package is to concentrate a great number of the declarations necessary for the implemented application program. Since, as already mentioned, the utilities programs coded in Janus/Ada have not yet all been delivered, some procedure to supply the parameters necessary for the create and makeknown calls, had to be built. For the time being, a simple loop that will generate fixed numbers is what will be used. A correct procedure, which will look for the next free number to allocate, should be done in the future. This package has a preliminary procedure to load access classes yet to be tested.

B. PROCESS STRUCTURE

1. Pathname Convention

Since all information in the Gemini system is stored in segments, some method to make reference to these segments is needed. A pathname is the shorthand method used for this purpose of aliasing a segment. It consists of a sequence of entry numbers that together define all of the mentor segments to a particular segment. The pathname "3,8" indicates that the target segment is at entry 8 of its mentor segment, which itself has entry 3 in the system mentor segment. Pathnames may be up to 5 entry numbers long in the present implementation.

The pathname is used during the generation of a program to run in the secure environment, and it will be explained in the next sections.

2. Ring 1 Environment

The current implementation of Janus/Ada on GEMSOS does not allow the use of a Janus/Ada process as the initial Ring 1 process. The Ring 1 Login and the Ring 1 Loader provided have to used in order to run Janus/Ada programs.

Another restriction imposed by this preliminary version is that, the file R1TRAP.CMD, which contains the trap handler and debugger, has to be sysgened (to be discussed later), at entry six off the system mentor segment. Figure 4.2 shows the Ring 1 environment segment naming hierarchy. The segments which have fixed "positions" in the ring 1 structure are shown in Figure 4.2, which are:

1) SSAT- System Segment Aliasing Table; containing the bootstrap and kernel segments.

2) V1loader- Ring 1 loader code segment.

3) V1logon- Login process code segment.

4) NV.DS at 5,0- Shared segment for loader processes.

5) NV.DS at 5- Appliccation Root

6) R1trap- Trap entry and debugger

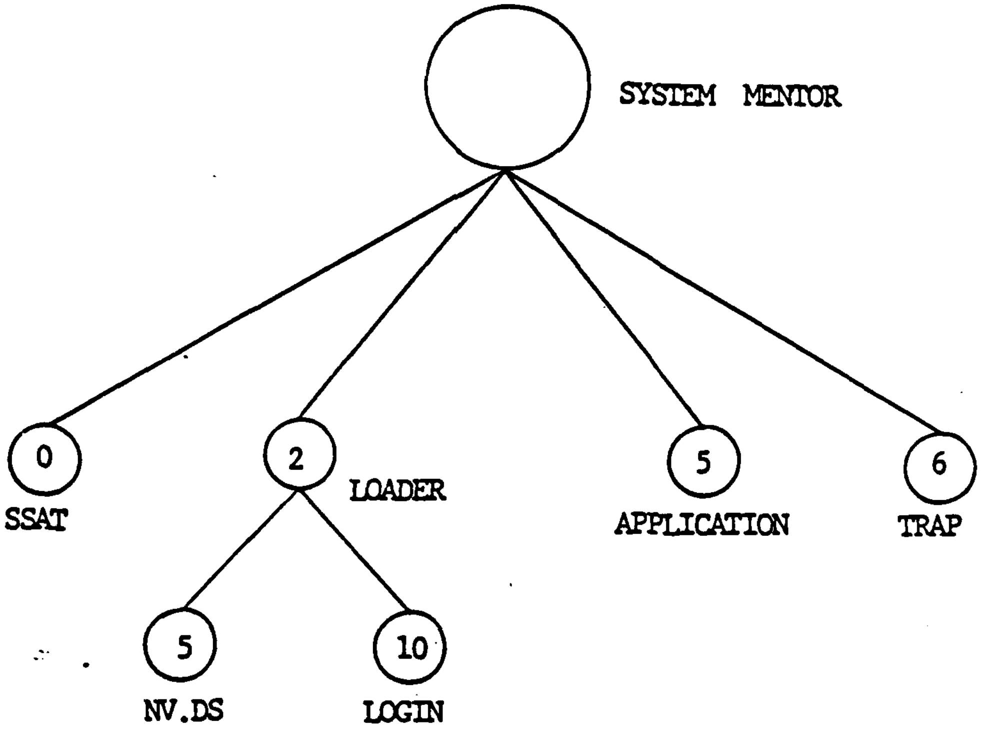


Figure 4.2 Ring 1 Structure

The entry of concern for the applications programmer, is entry number 5. In the current implementation, this is the position were the application program should be located.

3. Application Program Environment

The application program developed in this research, is composed of four segments. The mentor segment for the code segments (the application mentor) will be at entry 5 off the system mentor. The mentor segment for the stack and data segments will be at entry 5 off the application mentor. The parent process will be located at entry 0 off the application mentor. The child processes will be located at entries 7, 8 and 9 off the application mentor. This scheme can be better explained by the diagram in Figure 4.3.

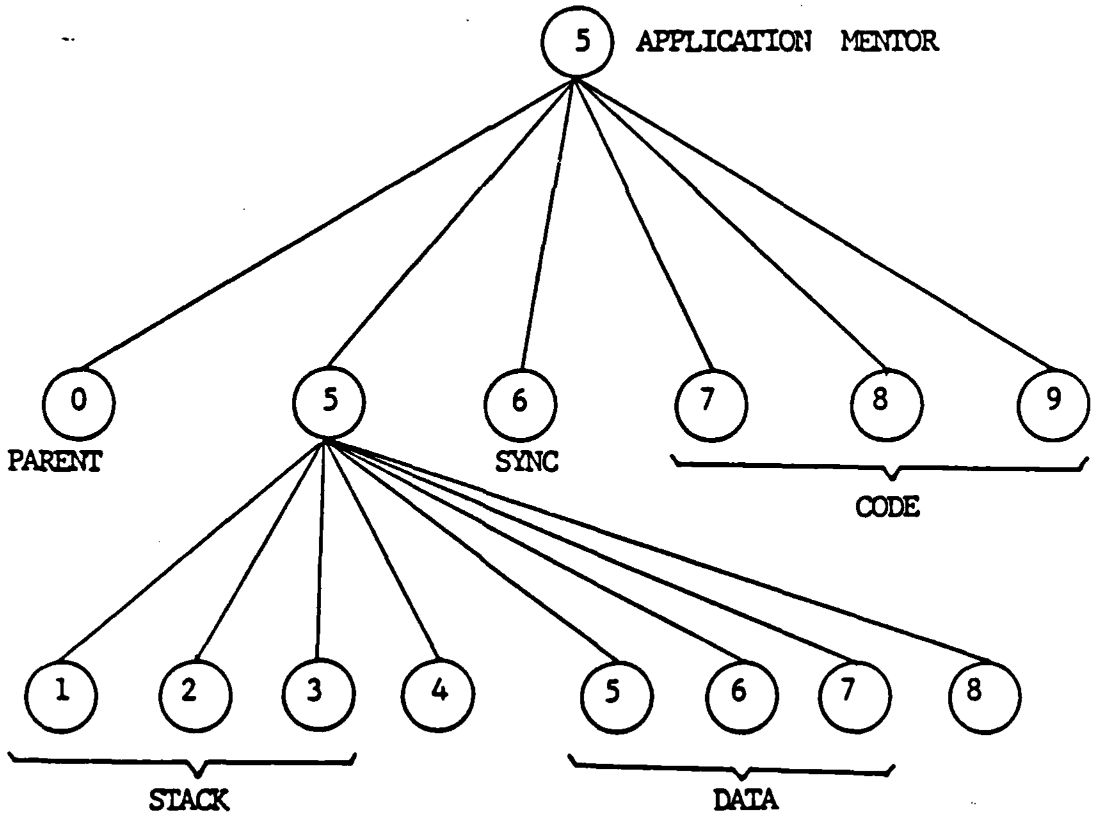


Figure 4.3 Application Program Structure

The entries used for this structure, were chosen with no special reason. Other combinations could have been chosen. The only restriction is to position the parent process at entry 0 off the application mentor.

These entry numbers are treated as paths, when referenced. So, entry 7 off entry 5 off the system mentor, is called 5,7. This will be used in the file with the commands for the sysgening phase. Those entries have to be passed as parameters during the creation and makeknown of segments.

For the application program developed, a segment for the 'purpose of executing the synchronization was created, and will be located at entry 6 off the application mentor.

4. Process Synchronization

Process synchronization is accomplished using the eventcount of the synchronization segment (5,6), and the eventcount of the stack segment of each child. Advancing each eventcount in turn, the parent process would prepare each child to execute its code, as soon as the parent makes an AWAIT call. The child, would then, after execution, advance the eventcount of the syncronization segment, which is the one being read by the parent process. This scheme would be repeated and the synchronization between all processes is achieved. The actual sequence of synchronization used in the programs developed, can be seen in the programs' listing in Appendix A.

C. GENERATION OF A SECURE PROGRAM

To prepare programs to run in the Gemini Secure Operating System (GEMSOS) environment is much more complicated than running a Janus/Ada program in a non-secure environment. There are some specific calls the program has to make, in order to be recognized by the GEMSOS, and gain access to the security kernel. The programs are compiled and linked like normal Janus/Ada programs. The command (CMD) files, will then be put in the secure environment. To assign the security classification and prepare the programs to run in a secure environment, a secure volume must be created by running the operating system generation program (SYSGEN). Execution of the SYSGEN program will include the application programs into a segment structure, which will then be transformed into a bootable executable program.

The SYSGEN program reads a submit file to identify the segment structure. This submit file, for the current implementation, will have the format as in Figure 4.4.

Except for the application.cmd and the child.cmd files, all the other segments are to be sysgened exactly as described in Figure 4.4. The submit file used to SYSGEN the application program implemented is listed in Appendix E .

```
File :  Application.ssb

bs:ld3.cmd
ks:k0.cmd
ks:k1.cmd
ks:k0h.cmd
ks:k2.cmd
cs:v1loader.cmd;2;
ds:v1login.cmd;2,10;
ds:nv.ds;2,5;
ds:nv.ds;5;
ds:application.cmd;5,0;
ds:child1.cmd;5,7;
ds:child2.cmd;5,8;
""""""""""""""""""""""""""""""""
""""""""""""""""""""""""""""""""
ds:r1trap.cmd;6;
end
```

Figure 4.4 Submit File

D. LOSS IN PERFORMANCE

In order to achieve a secure environment, we have developed our program using four different processes, which can have different access classes. During the execution of the secure program, the operating system will perform security checks each time a process is brought into execution and each time a segment is accessed. Evidently, some overhead, in comparison with a non-secure system, exists. A test program was developed in order to extract the preliminary measurements of such a overhead.

1. Test Program

The same structure used in the application process developed, was used in this test program.

The algorithm used in the main program is described in Figure 4.5.

```
Package body TOTIME is

begin
    perform initialization
    create parent segment
    create synchronization segment
    create processes
    case
       execute calculations with no calls
       execute calculations with one call
       execute calculations with calls
                      every loop
    end case
    delete processes
end TOTIME;
```

Figure 4.5 Package TIME

When the procedure to execute calculations with no calls is activated, the program will perform a simple arithmetic calculation 30000 times. These calculations will be performed by the main process, after the creation of all child processes, and with values already in the main process, so there are no GEMSOS calls.

When the procedure to perform calculations with one call is activated, the program will activate two processes: CALC1 and STODISP. The STODISP process will supply one value to the CALC1 process. CALC1 process will receive this value and perform the same arithmetic calculation as before 30000 times as well. The actual value passed by one process to the other is irrelevant, and it is there just to provoke a call to the operating system during the transmission of the value. The result of the calculation will be displayed by the STODISP process.

Finally, the procedure to perform calculations with call in every loop is activated. The program will then activate CALC2 and STODISP processes. Process CALC2 will perform the same calculations as before, but this time will include in every loop of the computation, a transmission of data between the STODISP and CALC2 segments.

Because a loop statement is being used to control these tests, two measurements are taken at the first step (calculations with no GEMSOS calls), in order to estimate the contribution of the loop control code to the overall time taken by the calculation.

All those steps are measured and analysed.

2. Performance Results

The results obtained from the test program are the following:

1) With no GEMSOS calls, 30000 operations => 3.33 seconds.

2) With four (4) GEMSOS calls, 30000 operations => 3.28 seconds.

3) With four (4) GEMSOS calls for each operation, 300 operations => 23.8 seconds.

At the first step, no GEMSOS calls, another measurement was taken, doubling the number of operations and maintaining the same loop number, in order to estimate the time delay contribution of the loop control. The time measured was 6.03 seconds, which shows that the actual operations take 2.7 seconds, and the loop control is responsible for 0.63 seconds. Since we execute the loop 30000 times, it is possible to estimate the time delay for each loop to be 21 μseconds. The times measured show that each mult/div operation, which there are 12 on each loop, is using 7.5 μseconds, which is the expectable time delay for a APX-286 CPU.

The time measured during the execution of the second step, the same number of operations plus four GEMSOS calls, did not show any appreciable difference.

At the third step, where there are four GEMSOS calls on each loop, and the loop is executed 300 times, the time measured was 23.8 seconds. As the test executes the loop 300 times, each loop uses 79 milliseconds. Assuming the same loop delay time as before, each loop control uses 21 μseconds, and the time delay of the actual calculations is (12 x 7.5 μseconds) 90 μseconds. Therefore, as four GEMSOS calls are executed in each loop, each call uses an average of 19 milliseconds.

These preliminary measurements are far from complete, but the results obtained can be considered as a design parameter to be expected when the security environment is used with the Gemini system.

V. APPRECIATION OF RESULTS

A. GENERAL COMMENTS

The development of application programs to execute in the Gemini microcomputer proved to be much more time consuming that it was anticipated. Testing and debugging of the programs could not be done using the techniques and skills normally used when working with non-secure systems. Some factors can be listed as the major ones which contributed to this problem. They were:

1) new terms and concepts that had to be completely understood before attempting to use the system

2) preliminary version of the manuals provided, which are still being updated and developed

3) preliminary version of the library programs which do not include yet, most of the common needed procedures

4) the system used was configured with two floppy disk drives.

The Janus/Ada gate calls for the Gemini Secure Operating System (GEMSOS), are not yet very well explained in the manuals provided. As such, any time a new call was to be tested, in order to increase our understanding of a new concept, the complete process of preparation of a program to run in a secure environment had to be executed. Since this process involves the access to a large number of files, the fact that the system used floppy disk drives, imposed a time delay of at least 7 minutes.

As discussed in Chapter III, in order to prepare a program to run in the secure environment, the operating system generation program (SYSGEN) had to be executed, which would create a secure volume containing the program segment. Before running the system generation program, the application program had to be compiled and linked in the normal way.

After the creation of the secure volume, the system has to be reinitialized with the secure application program volume. If a problem is found in the execution of the program, the system will either execute an interrupt trap halt indicating the processor's register contents, or sometimes will halt completely not giving any indication on the screen. The error then, must be corrected before any further progress can be achieved. After the correction has been made, the preparation process has to be repeated completely to check if the modification was successful. The average amount of time from compilation to the final run of a program, was found to be between 15 to 17 minutes for the application programs developed in this research.

The use of the modular programming technique is very important for the compilation and linking phases, but as the preparation of the secure program has the "sysgening" phase, where all the modules have to be included, the modularity does not bring great advantages to the preparation phase.

Improved versions of the Gemini system will certainly become available in the near future, which will reduce significantly the effects of these problems. System libraries will be expanded, making the process of writing programs to be run in the secure system less complicated and time consuming.

B. SYSTEM OPERATION

The system designed proved the possibility of using a secure computer system as the main building block in a tactical computer system. The ability of handling different and somewhat independent processes, easily synchronized by the calls already provided by the operating system, was demonstrated by the model implemented. The actual code of the modules used do not represent any real application, but only exemplifies that they can be independently developed, and integrated into a complete system.

Due to the problems already discussed in the previous sections, the amount of time for this research, was not sufficient to proceed with the next step scheduled, which was, to label one of the processes as secret and then limit the access based on the security level of the operator logged on the system.

The use of the system clock to control the synchronization between the different processes involved in the application program could not be completed in time to be included in this thesis.

The overhead analysed has shown that an average of 19 miliseconds is used for each GEMSOS call, where synchronization and security checks are performed. This time delay has to be taken into account when the tactical system is designed, but certainly it is not a high price to pay in order to be able to develop a system with a large number of security possibilities available.

C. CONCLUSIONS AND SUGGESTIONS

In this thesis, a model of a tactical combat system was developed to demonstrate the possibility of using a multilevel secure computer system in this environment. The Gemini Trusted Multiple Microcomputer Base used in this research, proved to be able to synchronize the execution of independent processes which will give the capability of assigning different security labels to these processes.

Although it has not been possible to achieve all the desired goals proposed when this thesis was first planned, the concepts and research done, will certainly facilitate any further work to be done in this new area.

Most of this research was done in conjunction with another student, Major Miguel Reyes, working with the same microcomputer, which to a certain extent guarantees that the results here obtained are completely known by a follow-up researcher.

The unit testing of application program modules should first be accomplished on development systems which have existing tools for testing the logical correctness and real time performance. A specially trained "lead programmer" should take the unit tested modules and incorporate these into a system's program which synchronizes the units and produces the necessary communications between the units in a secure systems environment. The art of systems integration programming in a secure environment requires an in-depth understanding of GEMSOS functions as well as the real-time performance of the system.

APPENDIX A

APPLICATION PROGRAM LISTING

This application program is compiled and prepared for execution in the manner discussed in Chapter III. The program consists of four packages, each one generating a separate command (CMD) file. The packages to be sysgened as child processes are designed to have procedures which can be altered without modifying the overall synchronization of the application program.

```
-----------------------------------------------------------
--   This package controls the operation of the complete --
--   system                                                --
-----------------------------------------------------------

with arl, alibj, agate, manag, tables, gemio, alib, crproce;
package body THEMAIN is
use  arl, alibj, agate, manag, tables, gemio, alib, crproce;

--  constants

STDIO_W : CONSTANT integer := 1;
STDIO_R : CONSTANT integer := 0;
IO_PORT : CONSTANT integer := 0; -- 0 port for main program

-- variables

init          :  rl_process_def;     --necessary for all kernel
calls
ch_table    : rl_param;
ch_level    : user_level;
seg_mode    : seg_access_type;
ch_tab      : rl_parameters;
ch_lev      : level_record;
w_class     : access_class;
class       : access_class;
rd_class    : access_class;
in_choice   : string;
pass_rad    : radar_input;
pass_chaf   : chaff_out;
mentor      : integer;
entryx      : integer;
def_seg     : integer;
def_off     : integer;
def_size    : integer;
size        : integer;
success     : integer;
seg_number  : integer;
synchr_seg  : integer;
choice      : integer;
evc_value   : integer;

procedure INITIALIZATION is

begin

-- attach serial port for writing

     attach_tew (IO_PORT, STDIO_W);

-- attach serial port for reading
```

```
     attach_ter (IO_PORT, STDIO_R);

-- load parameters to create up to 4 children

     load_param_to_4_chld(init, ch_table);

--load access classes for Top-Secret,  Secret,  Confidential
and  Unclassified .

     load_access_class (init, ch_level);

-- prepare class for accessing main terminal

     w_class := init.resources.min_class;

end INITIALIZATION;

procedure STACK_AND_SYNC_CREATION is

begin

-- creating segment for stack(parent).Will be unclassified
-- so as to obey compatibbility property : segment compromise
-- must dominate mentor compromise.

     mentor := init.initial_seg(2);
     entryx := 5;
     class := init.resources.min_class;
     size := 128;

     cr_segment(init,mentor,entryx,size,class,success);

     if success /= 0 then
          put_succ("success stack parent ",success,w_class);
          put_ln (STDIO_W,w_class,"");
     end if;

-- makeknown this segment

     seg_mode := r_w;
     seg_number := 31;

     mk_segment(init,mentor,entryx,
                            seg_number,seg_mode,success);

     if success /= 0 then

          put_succ("success  makeknown
                            stack  parent",success,  class);
          put_ln (STDIO_W,w_class,"");
     end if;

-- creating synchronization segment . Will be Top-Secret.

     mentor := init.initial_seg(2);
     entryx := 6;
     class := init.resources.min_class;

     cr_segment(init,mentor,entryx,size,class,success);

     if success /= 0 then
          put_succ("success sync is",success,w_class);
          put_ln(STDIO_W,w_class,"");
     end if;

-- makeknown this segment

     seg_mode := r_w;
     seg_number := 51;

mk_segment(init,mentor,entryx,seg_number,seg_mode,success);

     if success /= 0 then
          put_succ("success mkknown sync",success,w_class);
          put_ln(STDIO_W,w_class,"");
     end if;

     synchr_seg := seg_number;

-- swapin this segment

     swapin_segment(seg_number,success);

     if success /= 0 then
          put_succ("success swapin sync",success,w_class);
          put_ln(STDIO_W,w_class,"");
     end if;


end STACK_AND_SYNC_CREATION;

procedure PROCESS_CREATION is

begin
     put_ln(STDIO_W,w_class,"Begin Process Creation");
     type_any_key_to_continue(w_class);

-- start creating processes in the system
-- process 1  == >  Radar
-- process 2  == >  Compute
-- process 3  == >  Chaff
-- NOTE :
-- all processes with unclassified acess class
-- next  version to have process 3 changed to Top-Secret  in
order
--                 to access Secret data.

     for i in 1..3 loop

          ch_tab := ch_table(i);
          ch_lev := ch_level(4);

          to_create_process(init,ch_tab,
                               ch_lev,i,synchr_seg,success);
     end loop;

end PROCESS_CREATION ;

procedure MENU (selection : out integer) is

-- Present  option  to run tactical program  or  alter  data
field
-- Data field to secret in next version

begin
     put_ln(STDIO_W,w_class,"Run Tactical
                    Program == > < any   key >");
     put_ln(STDIO_W,w_class,"Alter
                    Data  Field   ==  >  <  A   >");
     put_ln(STDIO_W,w_class,"Exit  Program
                                  ==  >  <  E   >");

     get_str(STDIO_R,rd_class,in_choice);
     if in_choice = "a" or in_choice = "A" then
          selection := 1;
     elsif in_choice = "e" or in_choice = "E" then
          selection := 2;
     else
          selection := 3;
     end if;

end MENU;

procedure ALTER is

begin
     put_ln(STDIO_W,w_class,"Not implemented yet");

end ALTER;

procedure RECEIVE_FM_RADAR is
```

```
begin
     def_seg :=
        lib_mk_sel (ldt_table,ch_table(1).seg_number_data);
     def_off := 0;
     def_size := radar_input'size /8;
     move_bytes(def_seg,def_off,get_ss(),
                              pass_rad'ADDRESS,def_size);


end RECEIVE_FM_RADAR ;

procedure PASS_TO_COMPUTE is

begin
     def_seg :=
         lib_mk_sel(ldt_table,ch_table(2).seg_number_data);
     def_off := 0;
     def_size := radar_input'size /8;
     move_bytes
(get_ss(),pass_rad'ADDRESS,def_seg,def_off,def_size);

end PASS_TO_COMPUTE ;

procedure RECEIVE_FM_COMPUTE is

begin
     def_seg:=lib_mk_sel(lxdt_table,
                           ch_txable(2).segx_number_date)x;
     def_off :=  0;
     def_size := chaff_out'size /8;
     move_bytes(def_seg,def_off,get_ss(), pass_chaf'ADDRESS
                                      ,def_size);

end RECEIVE_FM_COMPUTE ;

procedure PASS_TO_CHAFF is

begin
     def_seg :=
           lib_mk_sel(ldt_table,ch_table(3).seg_number_data);
     def_off := 0;
     def_size := chaff_out'size /8;
     move_bytes(get_ss(),pass_chaf'ADDRESS,
                               def_seg,def_off,def_size);

end PASS_TO_CHAFF;


procedure RUN is
```

```
begin

   pass_rad.flag_z := false;

 outer : loop
    inner : for i in 1..3 loop
        advance(ch_table(i).seg_number_stack,success);
            read_evc(synchr_seg,evc_value,success);
            await(synchr_seg,evc_value+1,success);
            if i = 1 then
               receive_fm_radar;
               if pass_rad.flag_z then
                  exit outer;
               end if;
               pass_to_compute;
            elsif i = 2 then
               receive_fm_compute ;
               pass_to_chaff;
            end if;
         end loop inner;
      end  loop outer;

end RUN;

procedure SELF_DELETION is

begin
   for i in 1..3 loop
      advance(ch_table(i).seg_number_stack,success);
      read_evc(synchr_seg,evc_value,success);
      await(synchr_seg,evc_value+1,success);
   end loop;


end SELF_DELETION;

procedure DELETE_PROCESS_SEGMENT is

begin
  for i in 1..3 loop
      child_delete(i-1, success);
   terminate_segment(ch_table(i).seg_number_stack,success);
   terminate_segment(ch_table(i).seg_number_data,success);
   terminate_segment(ch_table(i).seg_number_code,success);
   delete_segment(ch_table(i).mentor_stack,i,success);
   delete_segment(ch_table(i).mentor_data,i+4,success);
   delete_segment(ch_table(i).mentor_code,i+6,success);
  end loop;


end DELETE_PROCESS_SEGMENT;
```

```
procedure DELETE_MENTOR_SYNC is

begin
     delete_segment (init.initial_seg(2), 6, success);
     terminate_segment (51, success);
     delete_segment (init.initial_seg(2), 5, success);
     terminate_segment (31 ,success);


end DELETE_MENTOR_SYNC ;


procedure DELETION_ALL is

begin
     self_deletion;
     delete_process_segment;
     delete_mentor_sync;

end DELETION_ALL ;

procedure PREVENT_TRAP is

begin
     success := 0;
     while success = 0 loop
          success := 0;
     end loop;
end PREVENT_TRAP ;

-- ############          MAIN      PROGRAM    #############

begin
     init := get_r1_def();
     lib_set_bracket(1,1,1,init.resources.min_class);
     initialization;
     stack_and_sync_creation;
     process_creation;
     loop
          menu(choice);
          case choice is
               when 1 => alter ;
               when 2 => EXIT;
               when 3 => run;
          end case;
     end loop;
     deletion_all;
     prevent_trap;
end THEMAIN ;
```

```
-----------------------------------------------------------
--  This package simulates the sampling of a tracker     --
--  radar, as an input to a tactical system              --
-----------------------------------------------------------
with arl,  manag,  gemio,  strlit,  agate ,  tables ,  alib,
alibj;
package body RADAR is
use  arl,  manag,  gemio,  strlit,  agate ,  tables ,  alib,
alibj;


-- constants

STDIO_W : CONSTANT integer := 1;
STDIO_R : CONSTANT integer := Ø;
IO_PORT : CONSTANT integer := 6;
INIT_DIST : CONSTANT integer := 1ØØØØ;
INIT_BEAR : CONSTANT integer := Ø9Ø;
CR         : CONSTANT integer := 13;
-- variables

init          : rl_process_def;
w_class       : access_class;
miss_rec      : radar_input;
success       : integer;
evc_ch_val    : integer;

def_seg       : integer;
def_off       : integer;
def_size      : integer;


procedure GET_TRACK is

-- simulate tracking of a missile
-- constants


begin
     miss_rec.radar1 := miss_rec.radar1 - 5Ø;
     if miss_rec.radar1 < 2ØØØ then
          miss_rec.radar2 := miss_rec.radar2 - 1;
     end if;
     put_str(STDIO_W,w_class, "RANGE ");
     put_dec(STDIO_W,w_class,miss_rec.radar1);
     put_str(STDIO_W,w_class,"     BEARING ");
     put_dec(STDIO_W,w_class,miss_rec.radar2);
     put_str(STDIO_W,w_class,char_to_str(character'val(CR)));
     if miss_rec.radar1 < 6ØØ then
          miss_rec.flag_z := true;
     end if;
end GET_TRACK;

procedure PASS_TO_PARENT is

begin
     def_seg := lib_mk_sel(ldt_table,init.initial_seg(3));
     def_off := 0;
     def_size := radar_input'size /8;
     move_bytes(get_ss(),miss_rec'ADDRESS,def_seg,def_off,def_size

end PASS_TO_PARENT ;

-- MAIN PROGRAM

begin
     init := get_r1_def();

-- attach terminal to write

     attach_tew(IO_PORT,STDIO_W);
     w_class := init.resources.min_class;

-- attach terminal to read

     attach_ter(IO_PORT,STDIO_R);

     put_ln(STDIO_W,w_class," R A D A R ");

-- Advance the eventcount of the synchronization segment
-- path 5,6 , plsn 51 , passed to child as ch_seg_list(2).
-- Will be recognized in child as init.initial_seg(2).

     advance(init.initial_seg(2),success);     -- this   will
                    permit creation of   processes   to go on

     read_evc(init.initial_seg(0),evc_ch_val,success);--
stack to sync
     await(init.initial_seg(0),evc_ch_val+1,success);
--   control sent  back to creation  of processes.

     miss_rec.flag_z := false;
     miss_rec.radar1 := INIT_DIST;
     miss_rec.radar2 := INIT_BEAR;

     loop
          get_track;             --  get track information
          pass_to_parent;
          advance(init.initial_seg(2),success);
          read_evc(init.initial_seg(0),evc_ch_val,success);
          await(init.initial_seg(0),evc_ch_val+1,success);
          if miss_rec.flag_z then
```

```
            miss_rec.radar1 := INIT_DIST;
            miss_rec.radar2 := INIT_BEAR;
        end if;

    end loop;

    advance(init.initial_seg(2),success);

-- detach and deletion

    detach_device(STDIO_R,success);
    detach_device(STDIO_W,success);
    self_delete(init.initial_seg(2),success);


end RADAR;
```

```
--------------------------------------------------------------
--      This package performs the actual computations      --
--------------------------------------------------------------

with ar1, manag, gemio, strlib, agate, tables, alib, alibj;
package body COMPUTE is
use  ar1, manag, gemio, strlib, agate, tables, alib, alibj;

-- constants

STDIO_W : CONSTANT integer := 1;
STDIO_R : CONSTANT integer := 0;
IO_PORT : CONSTANT integer := 3;
CR      : CONSTANT integer := 13;
-- variables

init          : r1_process_def;
w_class       : access_class;
rad_in        : radar_input;
cha_out       : chaff_out;
ship_rec      : ship_param;
def_seg       : integer;
def_off       : integer;
def_size      : integer;
success       : integer;
evc_ch_val    : integer;


procedure RECEIVE_FM_PARENT is

begin
     def_seg := lib_mk_sel(ldt_table, init.initial_seg(3));
     def_off := 0;
     def_size := radar_input'size /8;
     move_bytes(def_seg,def_off,
                          get_ss(),rad_in'ADDRESS,def_size);

end RECEIVE_FM_PARENT;

procedure PASS_TO_PARENT is

begin
     def_seg := lib_mk_sel(ldt_table, init.initial_seg(3));
     def_off := 0;
     def_size := chaff_out'size /8;
     move_bytes(get_ss(),cha_out'ADDRESS,
                                 def_seg,def_off,def_size);

end PASS_TO_PARENT ;

procedure CALCULATION is
```

```
begin
     put_str(STDIO_W,w_class," Computing ... ");
    put_str(STDIO_W,w_class,char_to_str(character'val(CR)));
     cha_out.chaff1 :=
                 ((rad_in.radar1/1000)*ship_rec.param1)+75;
     cha_out.chaff2 := (rad_in.radar2/10) + 30;

end CALCULATION ;


-- MAIN PROGRAM

begin
     ship_rec.param1 := 2;
     init := get_r1_def();

-- attach terminal to write

     attach_tew (IO_PORT,STDIO_W);
     w_class := init.resources.min_class;

--attach terminal to read

     attach_ter(IO_PORT,STDIO_R);

     put_ln(STDIO_W,w_class," C O M P U T E  ");

-- advance eventcount of synchro segment path 5,6 plsn 51
-- passed to child as ch_seg_list(2).
-- Will be called in child as init.initial_seg(2)

     advance (init.initial_seg(2),success);
     read_evc(init.initial_seg(0),evc_ch_val,success);
     await(init.initial_seg(0),evc_ch_val+1,success);

     cha_out.flag_z := false;

     loop
          receive_fm_parent;
          calculation;
          if rad_in.radar1 < 1500 then
               put_ln(STDIO_W,w_class,"");
               put_str(STDIO_W,w_class," F I R E ");
          end if;
          pass_to_parent;
          advance(init.initial_seg(2),success);
          read_evc(init.initial_seg(0),evc_ch_val,success);
          await(init.initial_seg(0),evc_ch_val+1,success);
     end loop;
     advance(init.initial_seg(2),success);

-- dettach and delete

     detach_device(STDIO_R, success);
     detach_device(STDIO_W,success);
     self_delete(init.initial_seg(2), success);

end COMPUTE ;
```

```
----------------------------------------------------------
-- This package simulates the driving of a weapon device --
----------------------------------------------------------

with arl, manag, gemio, agate ,strlib, tables, alib, alitj ;
package body CHAFF is
use  arl, manag, gemio, agate ,strlib, tables, alib, alitj ;

--constants

STDIO_W : CONSTANT integer := 1;
STDIO_R : CONSTANT integer := 0;
IO_PORT : CONSTANT integer := 5;
CR      : CONSTANT integer := 13;
-- variables

init          : r1_process_def;
w_class       : access_class;
cha_cont      : chaff_out;
def_seg       : integer;
def_off       : integer;
def_size      : integer;
success       : integer;
evc_ch_val    : integer;


procedure RECEIVE_FM_PARENT is

begin
     def_seg := lib_mk_sel(ldt_table,init.initial_seg(3));
     def_off := 0;
     def_size := chaff_out'size /8;
  move_bytes(def_seg,def_off,get_ss(),
                             cha_cont'ADDRESS,def_size);

end RECEIVE_FM_PARENT ;

--  MAIN PROGRAM

begin
     init := get_r1_def();

-- attach terminal to write

     attach_tew(IO_PORT,STDIO_W);
     w_class := init.resources.min_class;

--attach terminal to read

     attach_ter(IO_PORT,STDIO_R);
     put_ln(STDIO_W,w_class," C H A F F ");
```

```
-- advance eventcount of sync segment path 5,6 plsn 51
-- passed to child as ch_seg_list(2).
-- Will be called in child as init.initial_seg(2)

     advance (init.initial_seg(2),success);
     read_evc(init.initial_seg(0),evc_ch_val,success);
     await(init.initial_seg(0),evc_ch_val+1,success);

     cha_cont.flag_z := false;

     loop
          receive_fm_parent;
          put_str(STDIO_W,w_class," BEARING  ");
          put_dec(STDIO_W,w_class,cha_cont.chaff1);
          put_str(STDIO_W,w_class,"      ELEVATION ");
          put_dec(STDIO_W,w_class,cha_cont.chaff2);
   put_str(STDIO_W,w_class,char_to_str(character'val(CR)));
          advance(init.initial_seg(2) , success);
          read_evc(init.initial_seg(0),evc_ch_val,success);
          await(init.initial_seg(0),evc_ch_val+1,success);
          if cha_cont.flag_z then
               put_ln(STDIO_W,w_class,"");
               put_ln(STDIO_W,w_class," P A R K E D ");
               cha_cont.flag_z := false;
          end if;
     end loop;

     put_ln(STDIO_W,w_class,"");
     put_ln(STDIO_W,w_class," P A R K E D ");
     advance(init.initial_seg(2),success);

-- dettach and delete

     detach_device(STDIO_R,success);
     detach_device(STDIO_W,success);
     self_delete(init.initial_seg(2),success);

end CHAFF ;
```

APPENDIX B

LIBRARY PROGRAMS LISTING

These packages were built in order to concentrate all common procedures used for the application program developed in this research. They are far from complete, although they establish the organization necessary to develop secure applications programs. Some of the procedures included in this library were taken from the demonstration program supplied by the manufacturers of the Gemini computer.

```
------------------------------------------------------------
--   Specification for the MANAG package                 --
--   Contains procedures to handle segments              --
------------------------------------------------------------

with agate, agatej, ar1, util;
package MANAG is
use  agate, agatej, ar1, util;

procedure CR_SEGMENT (init   : in r1_process_def;
                      mentor : in integer;
                      entrx  : in integer;
                      size   : in integer;
                      class  : in access_class;
                      success: out integer );

procedure MK_SEGMENT (  init   : in r1_process_def;
                        mentor: in integer;
                        entrx : in integer;
                        number : in integer;
                        mode : in seg_access_type;
                        success : out integer );


procedure ATTACH_TEW( IO_PORT : in integer;
                         LDEV : in integer);

-- attach to write; IO_PORT is physical device ;
--                      LDEV    is logical device

procedure ATTACH_TER( IO_PORT : in integer;
                         LDEV : in integer) ;

-- attach to read;   IO_PORT is physical device
--                      LDEV is logical device

procedure b24_FRM_INTEGER (in_val     : in integer;
                           b24_val    : out b24_type );


end MANAG;
```

```
---------------------------------------------------------------
--  This package has procedures to handle segments           --
--  and terminals                                             --
--                                                            --
---------------------------------------------------------------

with agate,  agatej, ar1,  util;
package body MANAG is
use  agate,  agatej, ar1,  util;

   -- Constants for device slots.

STDIO_W : CONSTANT integer := 1;
STDIO_R : CONSTANT integer := Ø;
IO_PORT  :  CONSTANT  integer := Ø;    -- port zero for  main
                                                        process



procedure CR_SEGMENT( init     : in r1_process_def;
                      mentor   : in integer;
                      entrx    : in integer;
                      size     : in integer;
                      class    : in access_class;
                      success  : out integer ) is

--  Create segment call

cr_seg_str : create_seg_struct;

begin
    cr_seg_str.mentor := mentor;
    cr_seg_str.entryx := entrx;
    cr_seg_str.limit := size;
    cr_seg_str.class := class;

    create_segment( cr_seg_str, success );

end CR_SEGMENT;


procedure MK_SEGMENT ( init     : in r1_process_def;
                       mentor   : in integer;
                       entrx    : in integer;
                       number   : in integer;
                       mode     :    in    seg_access_type;
                       success  : out integer ) is

--  Makeknown segment call

seg_rec : mk_kn_struct;
```

```
seg_ret_rec : mk_kn_return;

begin
    seg_rec.mentor := mentor;
    seg_rec.entryx := entrx;
    seg_rec.seg_number := number;
    seg_rec.seg_mode := mode;
    seg_rec.prot_level  :=  byte(  1  );             --ring  1
                                                   protection
    seg_rec.gate_number := NULL_INDEX;       --no gate
    seg_rec.gate_prot := byte( 0 );

    makeknown_segment ( seg_rec, seg_ret_rec, success );

end MK_SEGMENT;


procedure ATTACH_TEW (IO_PORT : in integer ;
                        LDEV : in integer) is

--    attach serial port writing

mode     : attach_struct;
success  : integer;

begin
    mode.dev_name := siow;
    mode.siow_rec.dev_num := io_port; --physical device
    mode.siow_rec.dev_type := io;      --device itself to be
                                                        used
    mode.siow_rec.dev_id := LDEV;     --logical device
    mode.siow_rec.mr1   :=   byte(  16#04D#   );    --device
                                                configuration
    mode.siow_rec.mr2 := byte( 16#03E# );
    mode.siow_rec.io_mode := asrt_rts;
    attach_device( mode, success );

end ATTACH_TEW;


procedure ATTACH_TER( IO_PORT : in integer;
                        LDEV : in integer) is

--   attach serial port for reading.

mode_r   : attach_struct;
success  : integer;

begin
    mode_r.dev_name := sior;
    mode_r.sior_rec.dev_num := io_port;
    mode_r.sior_rec.dev_type := io;
    mode_r.sior_rec.dev_id := LDEV;
    mode_r.sior_rec.mr1 := byte( 16#Ø4D# );
    mode_r.sior_rec.mr2 := byte( 16#Ø3E# );
    mode_r.sior_rec.io_mode := asrt_dtr;
    mode_r.sior_rec.delim_active := FALSE;
    mode_r.sior_rec.delimiter := byte( 13 );
    mode_r.sior_rec.maximum   :=  1;   -- only   reads   one
                                           character at a time.
    attach_device( mode_r, success );

end ATTACH_TER;

procedure b24_FRM_INTEGER (in_val   : integer;
                           b24_val  : out b24_type) is

-- to convert an integer into a b24_type variable (3 bytes)
begin
    b24_val.byte2 := byte( Ø );
    b24_val.byte1 := hi( in_val );
    b24_val.byteØ := lo( in_val );

end b24_FRM_INTEGER;

end MANAG;
```

```
with agate, agatej, strlib ;
package GEMIO  is
use  agate, agatej, strlib ;


procedure PUT_LN (     ldev : in integer;
                    w_class : in access_class;
                        str : in string );


procedure GET_STR (    ldev : in integer;
                    r_class : out access_class;
                        str : out string );


procedure PUT_STR (    ldev : in integer;
                    w_class : in access_class;
                        str : in string );

procedure PUT_DEC (    ldev : in integer;
                    w_class : in access_class;
                       dval : in integer ) ;

procedure PUT_SUCC(  in_str : in string;
                    dec_val : in integer;
                    w_class : in access_class );

procedure TYPE_ANY_KEY_TO_CONTINUE (w_class : in
                                  access_class) ;

procedure BLK_SCR (    ldev : in integer;
                    w_class : in access_class);


end GEMIO;
```

```
----------------------------------------------------------------
--   This package contains procedure to handle I/O            --
----------------------------------------------------------------
with agate, agatej, strlit;
package body GEMIO is
use  agate, agatej, strlit;

STDIO_W : CONSTANT integer := 1;
STDIO_R : CONSTANT integer := 0;


procedure PUT_LN (    ldev : in integer;
                   w_class : in access_class;
                       str : in string ) is

--   put a string on device ldev with cr and lf

out_buf : string( 82 );
success : integer;
wt_sio : wt_seq_struct;
size_str : integer;
CR : CONSTANT integer := 13;
LF : CONSTANT integer := 10;

begin
    out_buf := str;
    size_str := length( str );
    out_buf := out_buf & char_to_str( character'val( CR ));
    out_buf := out_buf & char_to_str( character'val( LF ));
    wt_sio.device := ldev;
    wt_sio.data_off := out_buf'ADDRESS + 1;
    wt_sio.data_seg := get_ss();
    wt_sio.count := size_str + 2;
    wt_sio.class := w_class;
    write_sequential( wt_sio, success );

end PUT_LN;


procedure GET_STR (    ldev : in integer;
                    r_class : out access_class;
                        str : out string ) is

--   get a string from device ldev.

in_buf : string( 82 );
success : integer;
rd_sio : rd_seq_struct;
rd_ret : rd_seq_return;
size_str : integer;
```

```
BEGIN
    rd_sio.data_off := in_buf'ADDRESS + 1;
    rd_sio.device := ldev;
    rd_sio.data_seg := get_ss();
    read_sequential( rd_sio, rd_ret, success );
    in_buf( 0 ) := character'val( rd_ret.count );
    str := in_buf;
    r_class := rd_ret.class;

end GET_STR;


procedure PUT_STR (    ldev : in integer;
                    w_class : in access_class;
                        str : in string ) is

--   put a string on device ldev.

out_buf : string;
success : integer;
wt_sio : wt_seq_struct;
size_str : integer;

begin
    out_buf := str;
    size_str := length( str );
    wt_sio.device := ldev;
    wt_sio.data_off := out_buf'ADDRESS + 1;
    wt_sio.data_seg := get_ss();
    wt_sio.count := size_str;
    wt_sio.class := w_class;
    write_sequential( wt_sio, success );

end PUT_STR;


procedure PUT_DEC(    ldev : in integer;
                   w_class : in access_class;
                      dval : in integer ) is

--  put  the string equivalent of a integer on the  terminal
screen.

out_buf : string( 10 );

begin
    out_buf := Int_to_str( dval );
    put_str( ldev, w_class, out_buf );

end PUT_DEC;
```

```
procedure PUT_SUCC( in_str : in string;
                    dec_val : in integer;
                    w_class : in access_class ) is

--   print  a  string and an integer on device  attached  in
slot STDIO_W
--   (should be a serial terminal).

begin
    put_str( STDIO_W, w_class, in_str );
    put_dec( STDIO_W, w_class, dec_val );
    put_ln( STDIO_W, w_class, "" );

end PUT_SUCC;

procedure     TYPE_ANY_KEY_TO_CONTINUE(w_class     :      in
access_class) is

rd_str : string;
rd_class : access_class;

begin

  put_str (STDIO_W,w_class, " type any key to continue");
  get_str (STDIO_R,rd_class,rd_str);
  put_ln (STDIO_W, w_class,rd_str);

end TYPE_ANY_KEY_TO_CONTINUE ;


procedure BLK_SCR (   ldev : in integer;
                    w_class: in access_class) is

-- clear screen and home cursor

out_buf : string;
success :integer;
wt_sio : wt_seq_struct;
ESC : CONSTANT integer := 27;
F   : CONSTANT integer := 45;

begin
    out_buf :=char_to_str(character'val(ESC));
    out_buf :=out_buf & char_to_str(character'val(F));
    wt_sio.device := ldev;
    wt_sio.data_off:=out_buf'ADDRESS + 1;
    wt_sio.data_seg:=get_ss();
    wt_sio.count:=2;
    wt_sio.class:=w_class;
    write_sequential(wt_sio,success);

end BLK_SCR;

end GEMIO;
```

```
------------------------------------------------------------
--  This package contains declarations for the            --
--  application programs                                  --
------------------------------------------------------------

with agate, ar1;
package  TABLES is
use  agate, ar1;


MAX_PROC                  : CONSTANT integer := 4;
MAX_LEVELS                : CONSTANT integer := 4;
TOP_SECRET                : CONSTANT integer := 1;
SECRET                    : CONSTANT integer := 2;
CONFIDENTIAL              : CONSTANT integer := 3;
UNCLASSIFIED              : CONSTANT integer := 4;


type R1_PARAMETERS is record
   entry_stack      : integer;
   mentor_stack     : integer;
   seg_number_stack : integer;
   entry_code       : integer;
   mentor_code      : integer;
   seg_number_code  : integer;
   entry_data       : integer;
   mentor_data      : integer;
   seg_number_data  : integer;
   evn_count        : integer;
   evn_count_data   : integer;
end record;

type R1_PARAM is array (1..MAX_PROC ) of r1_parameters;


type LEVEL_RECORD is record
     min                  : access_class;
     max                  : access_class;
end record;

type USER_LEVEL is array (0..MAX_LEVELS) of level_record;


type SHIP_PARAM is record
     param1 : integer;
     param2 : integer;
     param3 : integer;
     flag_z : boolean;
end record;

type RADAR_INPUT is record
```

```
     radar1 : integer;
     radar2 : integer;
     radar3 : integer;
     flag_z : boolean;
end record;

type CHAFF_OUT is record
     chaff1 : integer;
     chaff2 : integer;
     chaff3 : integer;
     flag_z : boolean;
end record;

type TEST_MESSAGE is record
     rec1    : integer;
     rec2    : integer;
     result  : integer;
     flag    : boolean;
end record;


procedure   LOAD_PARAM_TO_4_CHLD   (        init    :    in
r1_process_def;
                                   ch_para : out r1_param);

procedure LOAD_ACCESS_CLASS(       init : in r1_process_def;
                             usr_access : out user_level);


end TABLES;
```

```
-----------------------------------------------------------
--  This package loads the parameters for the segments   --
--  Also loads security parameters                       --
-----------------------------------------------------------
with agate, ar1 ;
package body TABLES is
use  agate, ar1 ;


procedure LOAD_PARAM_TO_4_CHLD (   init : in r1_process_def;
                               ch_para : out r1_param) is


--  load the segments specifications

INITIAL   : CONSTANT integer := 31;
NEXT_NUMBER_FREE  : CONSTANT integer := 40;

prep : r1_parameters;

begin
    for i in 1..4 loop
        prep.entry_stack := i;
        prep.mentor_stack := INITIAL;
        prep.seg_number_stack := INITIAL + i;
        prep.seg_number_code := INITIAL + i + 4;
        prep.entry_code := i + 6;
        prep.mentor_code := init.initial_seg(2);
        prep.entry_data := i + 4;
        prep.mentor_data := INITIAL;
        prep.seg_number_data := NEXT_NUMBER_FREE + i - 1;

        ch_para(i) := prep;
    end loop;

end LOAD_PARAM_TO_4_CHLD ;


procedure LOAD_ACCESS_CLASS (      init : in r1_process_def;
                           usr_access : out user_level ) is
--
--  load user security levels
--
usr_level  : level_record;

BEGIN
    usr_level.min.compromise.int0 := 0;
    usr_level.min.compromise.int1 := 0;
    usr_level.min.integrity.int0  := 0;
    usr_level.min.integrity.int1  := 21504;
    usr_level.max.compromise.int0 := 6;
```

```
        usr_level.max.compromise.int1 := 0;
        usr_level.max.integrity.int0  := 0;
        usr_level.max.integrity.int1  := 21504;

        usr_access(TOP_SECRET) := usr_level;


        usr_level.min.compromise.int0     := 0;
        usr_level.min.compromise.int1     := 0;
        usr_level.min.integrity.int0      := 0;
        usr_level.min.integrity.int1      := 21504;
        usr_level.max.compromise.int0     := 4;
        usr_level.max.compromise.int1     := 0;
        usr_level.max.integrity.int0      := 0;
        usr_level.max.integrity.int1      := 21504;

        usr_access(SECRET) := usr_level;

        usr_level.min.compromise.int0 := 0;
        usr_level.min.compromise.int1 := 0;
        usr_level.min.integrity.int0  := 0;
        usr_level.min.integrity.int1  := 21504;
        usr_level.max.compromise.int0 := 2;
        usr_level.max.compromise.int1 := 0;
        usr_level.max.integrity.int0  := 0;
        usr_level.max.integrity.int1  := 21504;

        usr_access(CONFIDENTIAL) := usr_level;

        usr_level.min.compromise.int0 := 0;
        usr_level.min.compromise.int1 := 0;
        usr_level.min.integrity.int0  := 0;
        usr_level.min.integrity.int1  := 21504;
        usr_level.max.compromise.int0 := 0;
        usr_level.max.compromise.int1 := 0;
        usr_level.max.integrity.int0  := 0;
        usr_level.max.integrity.int1  := 21504;

        usr_access(UNCLASSIFIED) := usr_level;

    end LOAD_ACCESS_CLASS;

    end TABLES;
```

```
----------------------------------------------------------
--  Specification for the Create Process Package        --
----------------------------------------------------------

with agate, agatej, arl, alib, alibj, manag, gemio, tables;
package CRPROCE is
use  agate, agatej, arl, alib, alibj, manag, gemio, tables;



procedure FILL_INIT(      init : in r1_process_def;
                       ch_init : out r1_process_def;
                     ch_access : in level_record ) ;

procedure TO_CREATE_PROCESS(     init : in r1_process_def;
                             ch_para  : in r1_parameters;
                            ch_access : in level_record;
                              proces  : in integer;
                           synchr_seg : in integer;
                              success : out integer);



end CRPROCE;
```

```
------------------------------------------------------------
--  This package contains the procedure to create          --
--  processes                                              --
------------------------------------------------------------
with agate, agatej, ar1, alib, alibj, gemio, manag, tables;
package body CRPROCE is
use  agate, agatej, ar1, alib, alibj, gemio, manag, tables;

   -- Constants for device slots.

STDIO_W : CONSTANT integer := 1;
STDIO_R : CONSTANT integer := 0;
IO_PORT  :  CONSTANT  integer := 0;   -- port zero for  main
process


procedure FILL_INIT(       init : in r1_process_def;
                        ch_init : out r1_process_def;
                      ch_access : level_record ) is

--     fill  in  the  initial  process  record  of  a  child
process.
--     called by to_create_process

begin
    ch_init.cpu := init.cpu;
    ch_init.num_cpu := init.num_cpu;
    ch_init.num_kst := init.num_kst;
    ch_init.root_access := init.root_access;
    ch_init.s_seg := 3;
    ch_init.resources.priority  :=  init.resources.priority;
--same as parent.
    b24_frm_integer( 60, ch_init.resources.memory );
    ch_init.resources.processes := 2;
    ch_init.resources.segmnts := 100;
--
--   this will be modified with the specific access class of
--   each process
--
    ch_init.resources.min_class := ch_access.min;
    ch_init.resources.max_class := ch_access.max;
    ch_init.ring_num := byte( 1 );
    ch_init.sp2 := 0;

end FILL_INIT;


procedure TO_CREATE_PROCESS(        init : in r1_process_def;
                               ch_par    : in r1_parameters;
                               ch_access : in level_record;
```

```
                                proces   : in integer;
                             synchr_seg : in integer;
                                success : out integer ) is

--    process creation

chld_seg :  r1_seg_struct;     -- r1_addr_array for  child's
                                              segment
ch_init : r1_process_def;    -- r1_process_def for child
seg_rec : create_seg_struct; -- used to create stack segment
seg1_mkn  :  mk_kn_struct;      -- used to make known  stack
                                              segment
seg1_ret : mk_kn_return;
crt_rec : r1_cp_struct;      -- create process structure
ch_seg_list : seg_array;
w_class : access_class;
evc_value : integer;
stack_size : integer;
seg_mgr_bytes : integer;
def_off : integer;
def_seg : integer;
r1_def_size : integer;

-- constants for determining stack size

r1_stack_size : CONSTANT integer := 16#FFF#;
vect_size : CONSTANT integer := 4;

BEGIN
    w_class := ch_access.min;

    seg1_mkn.mentor := ch_par.mentor_code;
    seg1_mkn.entryx := ch_par.entry_code;
    seg1_mkn.seg_number := ch_par.seg_number_code;
    seg1_mkn.seg_mode := r_e;
    seg1_mkn.prot_level := byte( 1 );
    seg1_mkn.gate_number := NULL_INDEX;         -- no gate

    makeknown_segment( seg1_mkn, seg1_ret, success );

    if success /= 0 then
       put_succ("success value is ",success,w_class);
       put_ln(STDIO_W,w_class,"");
    end if;


--   address spec for child's stack

    chld_seg.seg_number := ch_par.seg_number_stack;
    chld_seg.seg_mode := r_w;
    chld_seg.swapin := TRUE;
    chld_seg.protect := byte( 1 );

    crt_rec.r1_addr_array( Ø ) := chld_seg;

--   address spec for child's code

    chld_seg.seg_number := ch_par.seg_number_code;
    chld_seg.seg_mode := r_e;
    chld_seg.swapin := TRUE;
    chld_seg.protect := byte( 1 );

    crt_rec.r1_addr_array( 1 ) := chld_seg;

--   address spec for child's mentor

    chld_seg.seg_number := synchr_seg;
    chld_seg.seg_mode := n_a;
    chld_seg.swapin := TRUE;
    chld_seg.protect := byte( 1 );

    crt_rec.r1_addr_array( 2 ) := chld_seg;
--
--   address spec for trap handler segment

    chld_seg.seg_number := init.initial_seg(4);
    chld_seg.seg_mode := r_e;
    chld_seg.swapin := TRUE;
    chld_seg.protect := byte( 1 );

    crt_rec.r1_addr_array( 4 ) := chld_seg;

--   address spec for child's data

    chld_seg.seg_number := ch_par.seg_number_data;
    chld_seg.seg_mode := r_w;
    chld_seg.swapin := TRUE;
    chld_seg.protect := byte( 1 );

    crt_rec.r1_addr_array( 3 ) := chld_seg;

--  fill the order in which the segments will be passed

    ch_seg_list(Ø)   :=  ch_par.seg_number_stack;
    ch_seg_list(1)   :=  ch_par.seg_number_code;
    ch_seg_list(2)   := synchr_seg;
    ch_seg_list(3)   := ch_par.seg_number_data;
    ch_seg_list(4)   := init.initial_seg(4);


--  calculate required stack size.
--  (in  the  future will calculate based on data  in  "CMD"
file header
--   but now just use constant.)

    seg_mgr_bytes := ( stack_header'SIZE/8 ) +
                      ( init.num_kst * ( kst_entry'SIZE/8 ))
                            +
                      ( kst_header'SIZE/8 );
    stack_size   := r1_stack_size + vect_size + seg_mgr_bytes
                            +
                      ( r1_process_def'SIZE/8 );

--     create and make known child's stack segment

    seg_rec.mentor := ch_par.mentor_stack;
    seg_rec.entryx := ch_par.entry_stack;
    seg_rec.limit := stack_size - 1;
    seg_rec.class := ch_access.min;

    create_segment( seg_rec, success );

    if success /= 0 then
       put_succ("success value chsta is ",success,w_class);
       put_ln(STDIO_W,w_class,"");
    end if;

    seg1_mkn.mentor := ch_par.mentor_stack;
    seg1_mkn.entryx := ch_par.entry_stack;
    seg1_mkn.seg_number := ch_par.seg_number_stack;
    seg1_mkn.seg_mode := r_w;
    seg1_mkn.prot_level := byte( 1 );
    seg1_mkn.gate_number := NULL_INDEX;
    seg1_mkn.gate_prot   := byte( 0 );

    makeknown_segment( seg1_mkn, seg1_ret, success );

    if success /= 0 then
       put_succ("success value mksta is ",success,w_class);
       put_ln(STDIO_W,w_class,"");
    end if;

    swapin_segment( ch_par.seg_number_stack, success );

    if success /= 0 then
       put_succ("success  value  swapsta  is
                                          ",success,w_class);
       put_ln(STDIO_W,w_class,"");
    end if;

    --      create and make known child's data segment

    seg_rec.mentor := ch_par.mentor_data;
```

```
          seg_rec.entryx := ch_par.entry_data;
          seg_rec.limit := test_message'size/8;
          seg_rec.class := ch_access.min;

          create_segment( seg_rec, success );

          if success /= 0 then
             put_succ("success value chdat is ",success,w_class);
             put_ln(STDIO_W,w_class,"");
          end if;

          seg1_mkn.mentor := ch_par.mentor_data;
          seg1_mkn.entryx := ch_par.entry_data;
          seg1_mkn.seg_number := ch_par.seg_number_data;
          seg1_mkn.seg_mode := r_w;
          seg1_mkn.prot_level := byte( 1 );
          seg1_mkn.gate_number := NULL_INDFX;
          seg1_mkn.gate_prot   := byte( 0 );

          makeknown_segment( seg1_mkn, seg1_ret, success );

          if success /= 0 then
             put_succ("success value mkdat is ",success,w_class);
             put_ln(STDIO_W,w_class,"");
          end if;

          swapin_segment( ch_par.seg_number_data, success );

          if success /= 0 then
             put_succ("success value swadat is ",success,w_class);
             put_ln(STDIO_W,w_class,"");
          end if;

      --  fill in childs r1_process_def

          fill_init( init, ch_init, ch_access );

      --  determine  segment and offset of r1_process_def  initial
      record

          def_seg  :=  lib_mk_sel(   ldt_table,
                                        ch_par.seg_number_stack );
          def_off := stack_size - ( vect_size + seg_mgr_bytes +
                         r1_process_def'SIZE/8 );

      --   move ch_init into proper place in child's stack segment

          r1_def_size := ( r1_process_def'SIZE )/8;
          move_bytes( get_ss(), ch_init'address, def_seg, def_off,
      r1_def_size );
```

```
--   fill in remainder of create_process_structure

    crt_rec.ip  :=  128;                              -- skip  command
                                                 file header (80 hex)
    crt_rec.spx := def_off;                         -- set childs stack
                                                             pointer
    crt_rec.sp1  := stack_size - ( vect_size + seg_mgr_bytes
                                                                  );
    crt_rec.sp2 := 0;                               -- no ring 2 stack
    crt_rec.vec_seg := 0;                           -- r1 address array
                                                           element 0
    crt_rec.vec_off := stack_size - vect_size;
    crt_rec.child_num                 :=                  proces-1;
    crt_rec.priority := ch_init.resources.priority;
    crt_rec.memory := ch_init.resources.memory;                    --
    crt_rec.processes := ch_init.resources.processes;
    crt_rec.segmnts := ch_init.resources.segmnts;
    crt_rec.min_class := ch_init.resources.min_class;
    crt_rec.max_class := ch_init.resources.max_class;

--   read event count so we prepare for synchronization

    read_evc(synchr_seg,evc_value, success );

--   create the process

    create_process( crt_rec, success );
    if success /= 0 THEN
        put_succ(  " create process success  =  ",  success,
                                                    w_class );
    end if;

    await(synchr_seg,  evc_value+1,  success );-- blocks and
                                                          await

                                          -- "goto" process created

end TO_CREATE_PROCESS ;
end CRPROCE;
```

APPENDIX C

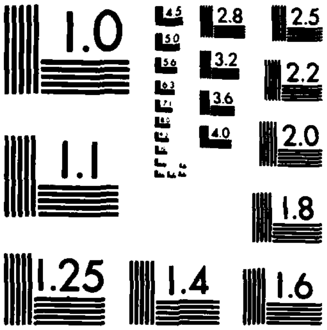
TEST PROGRAM LISTING

This program was developed following the general format of the application program in Appendix A. The preparation of this program to execute in the secure environment is done in the same way as the application program.

```
-----------------------------------------------------
--  This package controls the operation of the test    --
--  program                                            --
-----------------------------------------------------

with ar1,  alib, alibj, agate, strlib, manag, tables, gemio,
crproce;
package body TOTIME is
use  ar1,  alib, alibj, agate, strlib, manag, tables, gemio,
crproce;

--  constants

STDIO_W : CONSTANT integer :=,1;
STDIO_R : CONSTANT integer := Ø;
IO_PORT : CONSTANT integer := Ø; -- Ø port for main program
BFL     : CONSTANT integer := 7;
-- variables

init         :  r1_process_def;    --necessary for all kernel
calls
ch_table    : r1_param;
ch_level    : user_level;
seg_mode    : seg_access_type;
ch_tab      : r1_parameters;
ch_lev      : level_record;
w_class     : access_class;
class       : access_class;
rd_class    : access_class;
in_choice   : string;
test_rec    : test_message;
mentor      : integer;
entryx      : integer;
def_seg     : integer;
def_off     : integer;
def_size    : integer;
size        : integer;
success     : integer;
seg_number  : integer;
synchr_seg  : integer;
choice      : integer;
evc_value   : integer;

procedure INITIALIZATION is

begin

-- attach serial port for writing

     attach_tew (IO_PORT, STDIO_W);

-- attach serial port for reading
```

AD-A171 395 MODELLING OF A MULTILEVEL SECURE TACTICAL COMBAT COMPUTER SYSTEM(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA C B CAVALCANTI JUN 86 2/2

UNCLASSIFIED F/G 9/2 NL



1.0
4.5
5.0
5.6
6.3
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

```
     attach_ter (IO_PORT, STDIO_R);

-- load parameters to create up to 4 children

     load_param_to_4_chld(init, ch_table);

--load access classes for Top-Secret,  Secret,  Confidential
and  Unclassified .

     load_access_class (init, ch_level);

-- prepare class for accessing main terminal

     w_class := init.resources.min_class;

end INITIALIZATION;

procedure STACK_AND_SYNC_CREATION is

begin

-- creating segment for stack(parent).Will be unclassified
-- so as to obey compatibility property : segment compromise
-- must dominate mentor compromise.

     mentor := init.initial_seg(2);
     entryx := 5;
     class := init.resources.min_class;
     size := 128;

     cr_segment(init,mentor,entryx,size,class,success);

     if success /= 0 then
          put_succ("success stack parent ",success,w_class);
          put_ln (STDIO_W,w_class,"");
     end if;

-- makeknown this segment

     seg_mode := r_w;
     seg_number := 31;

mk_segment(init,mentor,entryx,seg_number,seg_mode,success);

     if success /= 0 then

          put_succ("success   makekenown   stack
                            parent",success, w_class);
          put_ln (STDIO_W,w_class,"");
     end if;
```

```
-- creating synchronization segment .

    mentor := init.initial_seg(2);
    entryx := 6;
    class := init.resources.min_class;

    cr_segment(init,mentor,entryx,size,class,success);

    if success /= 0 then
        put_succ("success sync is",success,w_class);
        put_ln(STDIO_W,w_class,"");
    end if;

-- makeknown this segment

    seg_mode := r_w;
    seg_number := 51;

mk_segment(init,mentor,entryx,seg_number,seg_mode,success);

    if success /= 0 then
        put_succ("success mkknown sync",success,w_class);
        put_ln(STDIO_W,w_class,"");
    end if;

    synchr_seg := seg_number;

-- swapin this segment

    swapin_segment(seg_number,success);

    if success /= 0 then
        put_succ("success swapin sync",success,w_class);
        put_ln(STDIO_W,w_class,"");
    end if;


end STACK_AND_SYNC_CREATION;

procedure PROCESS_CREATION is

begin
    put_ln(STDIO_W,w_class,"Begin Process Creation");
    type_any_key_to_continue(w_class);

-- start creating processes in the system
-- process 1  == >  Store and Display
-- process 2  == >  Calc1 -> one field passed
-- process 3  == >  Calc2 -> field passed every loop
```

```
     for i in 1..3 loop

          ch_tab := ch_table(i);
          ch_lev := ch_level(4);

          to_create_process(init,ch_tab,
                   ch_lev,i,synchr_seg,success);
     end loop;

end PROCESS_CREATION ;

procedure MENU (selection : out integer) is

-- Present option to execute each timing program

begin
     put_ln(STDIO_W,w_class,"Execute with no GEMSOS calls");
     put_ln(STDIO_W,w_class,"12 mult/div 30000 times");
     put_ln(STDIO_W,w_class,"                    => < 1 >");
     put_ln(STDIO_W,w_class,"Execute passing data 4 times");
     put_ln(STDIO_W,w_class,"12 mult/div , 30000 times");
     put_ln(STDIO_W,w_class,"                   => < 2 >");
     put_ln(STDIO_W,w_class,"Execute passing data p/ loop");
     put_ln(STDIO_W,w_class,"12 mult/div , 300   times");
     put_ln(STDIO_W,w_class,"                    => < 3 >");
     put_ln(STDIO_W,w_class,"Exit                => < 4 >");

     get_str(STDIO_R,rd_class,in_choice);
     if in_choice = "1" then
          selection := 1;
     elsif in_choice = "2" then
          selection := 2;
     elsif in_choice = "3" then
          selection := 3;
     elsif in_choice = "4" then
          selection := 4;
     else
          selection := 5;
     end if;

end MENU;

procedure START is

begin
     put_ln(STDIO_W,w_class," Prep to time ... ");
     type_any_key_to_continue (w_class);
end START;

procedure FINISH is
```

```
begin
     put_str(STDIO_W,w_class,
                          char_to_str(character'val(BEL)));
     put_ln(STDIO_W,w_class,"S T O P ");
     type_any_key_to_continue(w_class);
end FINISH;

procedure RECEIVE_FM_STO is

begin
     def_seg :=
        lib_mk_sel (ldt_table,ch_table(1).seg_number_data);
     def_off := 0;
     def_size := test_message'size /8;
     move_bytes(def_seg,def_off,get_ss(),
                              test_rec'ADDRESS,def_size);


end RECEIVE_FM_STO ;

procedure PASS_TO_CALC1 is

begin
     def_seg  :=
          lib_mk_sel(ldt_table,ch_table(2).seg_number_data);
     def_off := 0;
     def_size := test_message'size /8;
     move_bytes
(get_ss(),test_rec'ADDRESS,def_seg,def_off,def_size);

end PASS_TO_CALC1 ;

procedure RECEIVE_FM_CALC1 is

begin
     def_seg:=lib_mk_sel(ldt_table,
                           ch_table(2).seg_number_data);
     def_off :=  0;
     def_size := test_message'size /8;
     move_bytes(def_seg,def_off,get_ss(),
                              test_rec'ADDRESS,def_size);

end RECEIVE_FM_CALC1 ;

procedure PASS_TO_CALC2 is

begin
     def_seg    :=
        lib_mk_sel(ldt_table,ch_table(3).seg_number_data);
     def_off := 0;
     def_size := test_message'size /8;
     move_bytes
(get_ss(),test_rec'ADDRESS,def_seg,def_off,def_size);

end PASS_TO_CALC2 ;

procedure RECEIVE_FM_CALC2 is

begin
     def_seg:=lib_mk_sel(ldt_table,
                           ch_table(3).seg_number_data);
     def_off :=  0;
     def_size := test_message'size /8;
     move_bytes(def_seg,def_off,get_ss(),
                               test_rec'ADDRESS,def_size);

end RECEIVE_FM_CALC2 ;

procedure PASS_TO_STO is

begin
     def_seg   :=
        lib_mk_sel(ldt_table,ch_table(1).seg_number_data);
     def_off := 0;
     def_size := test_message'size /8;
     move_bytes(get_ss(),test_rec'ADDRESS,
                                def_seg,def_off,def_size);

end PASS_TO_STO ;


procedure CALC_NO_CALLS is

FIRST : CONSTANT integer := 10000;
SECOND: CONSTANT integer := 500;

begin
     start;
     for i in 1..30000 loop
          test_rec.result := (( 10000 / 500 ) * 300 ) / 100;
          test_rec.result := (( 10000 / 500 ) * 300 ) / 100;
          test_rec.result := (( 10000 / 500 ) * 300 ) / 100;
          test_rec.result := (( 10000 / 500 ) * 300 ) / 100;
     end loop;
     finish;
     put_ln(STDIO_W,w_class,"now   do  the  same   operation
twice");
     put_ln(STDIO_W,w_class,"calculate   the  loop   control
time");
    start;
```

```
     for i in 1..30000 loop
          test_rec.result := (( 10000 / 500 ) * 300 ) / 100;
          test_rec.result := (( 10000 / 500 ) * 300 ) / 100;
          test_rec.result := (( 10000 / 500 ) * 300 ) / 100;
          test_rec.result := (( 10000 / 500 ) * 300 ) / 100;
          test_rec.result := (( 10000 / 500 ) * 300 ) / 100;
          test_rec.result := (( 10000 / 500 ) * 300 ) / 100;
          test_rec.result := (( 10000 / 500 ) * 300 ) / 100;
          test_rec.result := (( 10000 / 500 ) * 300 ) / 100;
     end loop;
     finish;

end CALC_NO_CALLS;

procedure CALC_ONE_PASS is

begin
     start;
     advance(ch_table(1).seg_number_stack,success);
     read_evc(synchr_seg,evc_value,success);
     await(synchr_seg,evc_value+1,success);
     receive_fm_sto;
     pass_to_calc1;
     advance(ch_table(2).seg_number_stack,success);
     read_evc(synchr_seg,evc_value,success);
     await(synchr_seg,evc_value+1,success);
     receive_fm_calc1;
     pass_to_sto;
     finish;
     advance(ch_table(1).seg_number_stack,success);
     read_evc(synchr_seg,evc_value,success);
     await(synchr_seg,evc_value+1,success);

end CALC_ONE_PASS;


procedure CALC_PASS_ALL is

begin
     start;
     for i in 1..300 loop
          advance(ch_table(1).seg_number_stack,success);
          read_evc(synchr_seg,evc_value,success);
          await(synchr_seg,evc_value+1,success);
          receive_fm_sto;
          pass_to_calc2;
          advance(ch_table(3).seg_number_stack,success);
          read_evc(synchr_seg,evc_value,success);
          await(synchr_seg,evc_value+1,success);
          receive_fm_calc2;
          pass_to_sto;
     end loop;

     finish;

end CALC_PASS_ALL ;


procedure SELF_DELETION is

begin
     test_rec.flag := true;
     for i in 1..3 loop
          def_seg  :=
         lib_mk_sel(ldt_table,ch_table(i).seg_number_data);
          def_off := 0;
          def_size := test_message'size /8;
          move_bytes (def_seg,def_off,get_ss(),
                               test_rec'ADDRESS,def_size);
          advance(ch_table(i).seg_number_stack,success);
          read_evc(synchr_seg,evc_value,success);
          await(synchr_seg,evc_value+1,success);
          put_succ("self deleted ",i,w_class);
     end loop;


end SELF_DELETION;

procedure DELETE_PROCESS_SEGMENT is

begin
     for i in 1..3 loop
     child_delete(i-1, success);
    terminate_segment(ch_table(i).seg_number_stack,success);
    terminate_segment(ch_table(i).seg_number_data,success);
    terminate_segment(ch_table(i).seg_number_code,success);
    delete_segment(ch_table(i).mentor_stack,i,success);
    delete_segment(ch_table(i).mentor_data,i+4,success);
    delete_segment(ch_table(i).mentor_code,i+6,success);
    put_succ("deleted ",i,w_class);
     end loop;


end DELETE_PROCESS_SEGMENT;

procedure DELETE_MENTOR_SYNC is

begin
     delete_segment (init.initial_seg(2), 6, success);
     terminate_segment (51, success);
     delete_segment (init.initial_seg(2), 5, success);
     terminate_segment (31 ,success);
```

```
end DELETE_MENTOR_SYNC ;


procedure DELETION_ALL is

begin
     self_deletion;
     delete_process_segment;
     delete_mentor_sync;
     put_ln(STDIO_W,w_class," O. K. ");
end DELETION_ALL ;


procedure PREVENT_TRAP is

begin
     success := 0;
     while success = 0 loop
          success := 0;
     end loop;

end PREVENT_TRAP ;

-- ########       MAIN             PROGRAM  ################

begin
     init := get_rl_def();  --must be the first statement
     lib_set_bracket(1,1,1,init.resources.min_class);
     initialization;
     stack_and_sync_creation;
     process_creation;
     test_rec.flag := false;
     loop
          menu(choice);
          case choice is
               when 1 => calc_no_calls;
               when 2 => calc_one_pass;
               when 3 => calc_pass_all;
               when 4 => exit;
               when 5 => null;
          end case;
     end loop;
     deletion_all;
     prevent_trap;

end TOTIME ;
```

```
----------------------------------------------------------------
-- This Package simulates the store process in the test     --
-- program                                                    --
----------------------------------------------------------------

with ar1, manag, gemio, agate, tables, alib, alibj;
package body STODISP is
use  ar1, manag, gemio, agate, tables, alib, alibj;

-- constants

STDIO_W : CONSTANT integer := 1;
STDIO_R : CONSTANT integer := 0;
IO_PORT : CONSTANT integer := 3;

-- variables

init          : r1_process_def;
w_class       : access_class;
test_rec      : test_message;
def_seg       : integer;
def_off       : integer;
def_size      : integer;
success       : integer;
evc_ch_val    : integer;


procedure RECEIVE_FM_PARENT is

begin
     def_seg := lib_mk_sel(ldt_table, init.initial_seg(3));
     def_off := 0;
     def_size := test_message'size /8;
     move_bytes(def_seg,def_off,get_ss(),test_rec'ADDRESS,
def_size);

end RECEIVE_FM_PARENT;

procedure PASS_TO_PARENT is

begin
     def_seg := lib_mk_sel(ldt_table, init.initial_seg(3));
     def_off := 0;
     def_size := test_message'size /8;
     move_bytes(get_ss(),test_rec'ADDRESS,def_seg,def_off,
def_size);

end PASS_TO_PARENT ;
```

```
-- MAIN PROGRAM

begin
     init := get_rl_def();

-- attach terminal to write

     attach_tew (IO_PORT,STDIO_W);
     w_class := init.resources.min_class;

--attach terminal to read

     attach_ter(IO_PORT,STDIO_R);

     put_ln(STDIO_W,w_class," STORAGE AND DISPLAY READY  ");

-- advance eventcount of synchro segment path 5,6 plsn 51
-- passed to child as ch_seg_list(2).
-- Will be called in child as init.initial_seg(2)

     advance (init.initial_seg(2),success);
     read_evc(init.initial_seg(0),evc_ch_val,success);
     await(init.initial_seg(0),evc_ch_val+1,success);


     loop
          pass_to_parent;
          advance(init.initial_seg(2),success);
          read_evc(init.initial_seg(0),evc_ch_val,success);
          await(init.initial_seg(0),evc_ch_val+1,success);
          receive_fm_parent;
          advance(init.initial_seg(2),success);
          read_evc(init.initial_seg(0),evc_ch_val,success);
          await(init.initial_seg(0),evc_ch_val+1,success);
          receive_fm_parent;
          if test_rec.flag then
               exit;
          end if;
     end loop;

     advance(init.initial_seg(2),success);

-- dettach and delete

     detach_device(STDIO_R, success);
     detach_device(STDIO_W,success);
     self_delete(init.initial_seg(2), success);

end STODISP ;
```

```
------------------------------------------------------------
-- This package performs one of the timing tests          --
------------------------------------------------------------

with ar1, manag, gemio, agate, tables, alib, alibj;
package body CALC1 is
use  ar1, manag, gemio, agate, tables, alib, alibj;

-- constants

STDIO_W : CONSTANT integer := 1;
STDIO_R : CONSTANT integer := 0;
IO_PORT : CONSTANT integer := 5;

-- variables

init         : r1_process_def;
w_class      : access_class;
test_rec     : test_message;
def_seg      : integer;
def_off      : integer;
def_size     : integer;
success      : integer;
evc_ch_val   : integer;


procedure RECEIVE_FM_PARENT is

begin
     def_seg := lib_mk_sel(ldt_table, init.initial_seg(3));
     def_off := 0;
     def_size := test_message'size /8;
     move_bytes(def_seg,def_off,get_ss(),
                                  test_rec'ADDRESS,def_size);

end RECEIVE_FM_PARENT;

procedure PASS_TO_PARENT is

begin
     def_seg := lib_mk_sel(ldt_table, init.initial_seg(3));
     def_off := 0;
     def_size := test_message'size /8;
     move_bytes(get_ss(),test_rec'ADDRESS,
                                def_seg,def_off,def_size);

end PASS_TO_PARENT ;

-- MAIN PROGRAM
```

```
begin
     init := get_r1_def();

-- attach terminal to write

     attach_tew (IO_PORT,STDIO_W);
     w_class := init.resources.min_class;

--attach terminal to read

     attach_ter(IO_PORT,STDIO_R);

     put_ln(STDIO_W,w_class," CALC ONE PASS READY ");

-- advance eventcount of synchro segment path 5,6 plsn 51
-- passed to child as ch_seg_list(2).
-- Will be called in child as init.initial_seg(2)

     advance (init.initial_seg(2),success);
     read_evc(init.initial_seg(0),evc_ch_val,success);
     await(init.initial_seg(0),evc_ch_val+1,success);


     loop
          receive_fm_parent;
          if test_rec.flag then
                exit;
          end if;

          for i in 1..30000 loop
         test_rec.result := (( 10000 / 500 ) * 300 ) /  100;
         test_rec.result := (( 10000 / 500 ) * 300 ) /  100;
         test_rec.result := (( 10000 / 500 ) * 300 ) /  100;
         test_rec.result := (( 10000 / 500 ) * 300 ) /  100;
          end loop;

          pass_to_parent;
          advance(init.initial_seg(2),success);
          read_evc(init.initial_seg(0),evc_ch_val,success);
          await(init.initial_seg(0),evc_ch_val+1,success);
     end loop;

     advance(init.initial_seg(2),success);

-- dettach and delete

     detach_device(STDIO_R, success);
     detach_device(STDIO_W,success);
     self_delete(init.initial_seg(2), success);

end CALC1 ;
```

```
with arl, manag, gemio, agate , tables , alib, alibj;
package body CALC2 is
use  arl, manag, gemio, agate , tables , alib, alibj;


-- constants

STDIO_W : CONSTANT integer := 1;
STDIO_R : CONSTANT integer := 0;
IO_PORT : CONSTANT integer := 6;

-- variables

init        : rl_process_def;
w_class     : access_class;
test_rec    : test_message;
success     : integer;
evc_ch_val  : integer;

def_seg     : integer;
def_off     : integer;
def_size    : integer;



procedure PASS_TO_PARENT is

begin
     def_seg := lib_mk_sel(ldt_table,init.initial_seg(3));
     def_off := 0;
     def_size := test_message'size /8;
     move_bytes(get_ss(),test_rec'ADDRESS,
                                    def_seg,def_off,def_size);

end PASS_TO_PARENT ;


procedure RECEIVE_FM_PARENT is

begin
     def_seg := lib_mk_sel(ldt_table, init.initial_seg(3));
     def_off := 0;
     def_size := test_message'size /8;
     move_bytes(def_seg,def_off,get_ss(),
                                      test_rec'ADDRESS,def_size);

end RECEIVE_FM_PARENT;


-- MAIN PROGRAM
```

```
begin
     init := get_r1_def();

-- attach terminal to write

     attach_tew(IO_PORT,STDIO_W);
     w_class := init.resources.min_class;

-- attach terminal to read

     attach_ter(IO_PORT,STDIO_R);

   put_ln(STDIO_W,w_class," CALC2 PASS EVERY LOOP READY");

-- Advance the eventcount of the synchronization segment
-- path 5,6 , plsn 51 , passed to child as ch_seg_list(2).
-- Will be recognized in child as init.initial_seg(2).

     advance(init.initial_seg(2),success);  -- this will
   permit creation of   processesto go on

     read_evc(init.initial_seg(0),evc_ch_val,success);
                                         --  stack to sync
     await(init.initial_seg(0),evc_ch_val+1,success);
  --   control sent  back to  creation  of processes

     loop
          receive_fm_parent;
          if test_rec.flag then
               exit;
          end if;
          test_rec.result := (( 10000 / 500 ) * 300 ) / 100;
          test_rec.result := (( 10000 / 500 ) * 300 ) / 100;
          test_rec.result := (( 10000 / 500 ) * 300 ) / 100;
          test_rec.result := (( 10000 / 500 ) * 300 ) / 100;
          pass_to_parent;
          advance(init.initial_seg(2),success);
          read_evc(init.initial_seg(0),evc_ch_val,success);
          await(init.initial_seg(0),evc_ch_val+1,success);


     end loop;

     advance(init.initial_seg(2),success);

-- detach and deletion

     detach_device(STDIO_R,success);
     detach_device(STDIO_W,success);
     self_delete(init.initial_seg(2),success);
```

```
end CALC2;
```

APPENDIX D

SIMPLE ACCESS PROGRAM LISTING

This program presents a very simple program, with the purpose to show the basic steps necessary to be able to access the secure system. Different from non-secure systems, the terminal is not automatically a part of the system, and as shown in this program, a GEMSOS gate call is necessary to include a terminal in the system.

```
------------------------------------------------------------
--  Sample program to access the system                   --
------------------------------------------------------------


pragma  rangecheck(  off  );  pragma debug(  off  );  pragma
arithcheck( off );
pragma enumtab( off );

WITH agate, ar1, alibj, util, manag ,gemio;
PACKAGE BODY alo IS
USE agate, ar1, alibj, util, manag ,gemio ;


       -- Constants for device slots.

STDIO_W : CONSTANT integer := 1;
STDIO_R : CONSTANT integer := 0;
IO_PORT  :  CONSTANT integer := 0;   -- port zero  for  main
process


-- Variables used by main program.
w_class : access_class;  -- AGATE
init : r1_process_def; -- AR1
mentor : integer ;
entrx : integer ;
size : integer ;
success : integer;
class : access_class ;
seg_mode : seg_access_type ; --AGATE
seg_number : integer ;
-- MAIN
BEGIN
    init := get_r1_def(); -- AR1
    lib_set_tracket( 1, 1, 1, init.resources.min_class );

--   attach serial port for writing.

    attach_tew( IO_PORT, STDIO_W ); --MANAG
    w_class := init.resources.min_class;

    put_ln(stdio_w,w_class, " HELLO COMPLICATED WORLD");

--  attach serial port for reading.

    attach_ter( IO_PORT, stdio_r); -- MANAG

    put_ln(stdio_w,w_class ,"now I will create a segment");
    type_any_key_to_continue (w_class);
```

```
--
--
-- creating segment for STACK (parent)

   mentor := init.initial_seg(2);
   entrx :=5;
   size :=1023;
   class := init.resources.min_class;

   cr_segment(init, mentor, entrx, size, class, success);


   put_ln(stdio_w,w_class,"now   I  will
                          make  the  segment known ");
   type_any_key_to_continue(w_class);

-- makeknown segment created

  seg_mode := r_w;
  seg_number := 31;

mk_segment(init, mentor, entrx,seg_number,seg_mode,success);

  put_ln(stdio_w,w_class," Ate logo (good bye)");

-- infinite loop to prevent trap.
  success := 0;
  while success = 0 loop
     success := 0;
  end loop;


end alo;
```

APPENDIX E

SUBMIT FILES LISTING

This appendix presents the submit files used for the sysgening of the application program, the testing program and the simple access program.

```
----------------------------------------------------------
--   SUBMIT FILE  FOR  APPLICATION PROGRAM              --
----------------------------------------------------------

ts:ld3.cmd
ks:k0.cmd
ks:k1.cmd
ks:k0h.cmd
ks:k2.cmd
cs:v1loader.cmd;2;
ds:v1login.cmd;2,10;
ds:nv.ds;2,5;
ds:nv.ds;5;
ds:themain.cmd;5,0;
ds:radar.cmd;5,7;
ds:compute.cmd;5,8;
ds:chaff.cmd;5,9;
ds:r1trap.cmd;6;
end
```

```
------------------------------------------------------------
--  SUBMIT FILE FOR TEST PROGRAM                          --
------------------------------------------------------------

ts:ld3.cmd
ks:k0.cmd
ks:k1.cmd
ks:k0h.cmd
ks:k2.cmd
cs:v1loader.cmd;2;
ds:v1login.cmd;2,10;
ds:nv.ds;2,5;
ds:nv.ds;5;
ds:totime.cmd;5,0;
ds:stodisp.cmd;5,7;
ds:calc1.cmd;5,8;
ds:calc2.cmd;5,9;
ds:r1trap.cmd;6;
end
```

```
-------------------------------------------------------------
--  SUBMIT FILE FOR SAMPLE PROGRAM                         --
-------------------------------------------------------------

bs:ld3.cmd
ks:kØ.cmd
ks:k1.cmd
ks:kØh.cmd
ks:k2.cmd
cs:v1loader.cmd;2;
ds:v1login.cmd;2,1Ø;
ds:nv.ds;2,5;
ds:nv.ds;5;
ds:alo.cmd;5,Ø;
ds:r1trap.cmd;6;
end
```

LIST OF REFERENCES

1. Allworth, S., Introduction to Real-Time Software Design, Springer-Verlag New York Inc., New York, 1981.

2. Department of Defense Computer Security Center, Ft. Meade, Md., Report CSC-STD-001-83, DOD TRUSTED COMPUTER SYSTEM EVALUATION CRITERIA, 15 August 1985.

3. Ames, S., Gasser, M., Schell, R, "Security Kernel Design and Implementation: An Introduction," Computer, v. 16, no. 7, July 1983.

4. Gemini Computers, Inc., Carmel, Ca., System Overview-Gemini Trusted Multiple Microcomputer Base, 11 May 1984.

5. Gemini Computers, Inc., Carmel, Ca., GEMSOS Ring 0 User's Manual for the Janus/Ada Language, December 1985.

6. Boebert, E., Kain, R., Young, B., "Trojan horse rolls up to DP gate," Computerworld, 2 December 1985.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943	1
4.	Dr. Uno R. Kodres, Code 52Kr Department of Computer Science Naval Postgraduate School Monterey, California 93943	3
5.	Lt. Philip J. Corbett, USN 72 Pilgrim Rd. Concord, Massachusetts 01742	1
6.	Daniel Green, Code 20F Naval Surface Weapons Center Dahlgren, Virginia 22449	1
7.	Capt. J. Donegan, USN PMS 400BS Naval Sea Systems Command Washington, D. C. 20362	1
8.	RCA AEGIS Data Repository RCA Corporation Government Systems Division Mail Stop 127-327 Moorestown, N. J. 08057	1
9.	Library (Code E33-05) Naval Surface Weapons Center Dahlgren, Virginia 22449	1

10. Dr. M. J. Gralia 1
Applied Physics Laboratory
Johns Hopkins Road
Laurel, Maryland 20707

11 Dana Small, Code 8242 1
NOSC
San Diego, California 92152

12. LCDR Claudio Bailly Cavalcanti, Brazilian Navy 2
Brazilian Naval Commission
4706 Wisconsin Ave. N. W.
Washington, D. C. 20016

13. CDR G. S. Baker, Code 52Bj 1
Department of Computer Science
Naval Postgraduate School
Monterey, California 93944

END

10-86

DTIC